ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΛΟΠΟΝΝΗΣΟΥ
ΤΜΗΜΑ ΕΠΙΣΤΗΜΗΣ ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ ΥΠΟΛΟΓΙΣΤΩΝ
ΜΕΤΑΠΤΥΧΙΑΚΟ ΠΡΟΓΡΑΜΜΑ ΣΠΟΥΔΩΝ 2010 – 2011

ΔΙΕΡΜΗΝΕΥΤΙΚΟΣ ΠΡΟΣΟΜΟΙΩΤΗΣ ΓΙΑ ΚΕΙΜΕΝΙΚΗ ΕΝΔΙΑΜΕΣΗ ΑΝΑΠΑΡΑΣΤΑΣΗ ΜΕΤΑΓΛΩΤΤΙΣΤΗ

«Interpretive Simulator for a Textual Compiler Intermediate Representation»

ΕΡΓΑΣΙΑ ΔΙΠΛΩΜΑΤΟΣ ΕΙΔΙΚΕΥΣΗΣ

Φοιτητής
Κανάτσος Άγγελος
pcst0911

Επιβλέπων Καθηγητής
Μασσέλος Κωνσταντίνος

Συνεπιβλέπων Καθηγητής
Καββαδίας Νικόλαος

Δεκέμβριος 2011

Στην Οικογένεια μου και στη Θεοδώρα…

Abstract

The intermediate code representation (IR) is consider as an important integral part of the compiler since it is reused across all important compilation stages for representing a source program; it connects the frontend to the backend of the compiler. Of the most popular representations used today are linearized forms such as TAC (Three Address Code). In recent years the scientific community has shown great interest in the development and improvement of representations of this kind for production use, as it is evident in popular compilation projects (GCC GIMPLE, LLVM, LANCE compiler IR-C). In this work, the intermediate language NAC (N-Address Code) is presented as an extended version of TAC. Instead of using three addresses in each IR statement proposal there may be N, theoretically infinite and practically limited only by the resources of the machine. NAC has been developed by Nikolaos Kavvadias [1] and for its simulation; a simulator interpreter named NacVM was developed that comprises the main topic of this work. The simulator was developed in ANSI C language and this first version suffices to support to a large extent the requirements posed by NAC.

Περίληψη

Η ενδιάμεση αναπαράσταση κώδικα ενός μεταγλωττιστή αποτελεί θα λέγαμε το σημαντικότερο στάδιο μεταγλώττισης ενός πηγαίου προγράμματος καθώς είναι εκείνο που ενώνει το frontend με το backend του μεταγλωττιστή. Τρία είναι τα είδη των αναπαραστάσεων που χρησιμοποιούνται σήμερα, με δημοφιλέστερο τον κώδικα TAC (Three Address Code). Τα τελευταία χρόνια η επιστημονική κοινότητα έχει δείξει μεγάλο ενδιαφέρον για την εξέλιξη και τη βελτίωση της μορφής αυτής, αφού παγκοσμίως αναπτύσσονται διάφορα projects (GCC GIMPLE, LLVM, LANCE compiler IR-C) με αυτό το σκοπό. Η γλώσσα NAC που θα παρουσιάσουμε αποτελεί μία επεκτεταμένη μορφή του TAC καθώς αντί για τρεις διευθύνσεις σε κάθε πρότασή της μπορούν να υπάρχουν N, δηλαδή απεριόριστες θεωρητικά (περιορίζεται από τους πόρους της μηχανής). Αναπτύχθηκε από τον Καββαδία Νικόλαο [1] και για την προσομοίωση της αναπτύξαμε ένα διερμηνευτικό προσομοιωτή με το όνομα NacVM ο οποίος θα αποτελέσει και το κύριο μέρος ανάλυσης αυτής της εργασίας. Αναπτύχθηκε σε γλώσσα ANSI C και η πρώτη έκδοση του ικανοποιεί σε μεγάλο βαθμό τις απαιτήσεις της NAC.

Πίνακας Περιεχομένων

Πίνακας Εικόνων

1. Εισαγωγή

Ο μεταγλωττιστής (compiler) είναι ένα πρόγραμμα το οποίο δέχεται σαν είσοδο το κώδικα ενός προγράμματος, γραμμένο σε μια γλώσσα Α και παράγει ως έξοδο ένα ισοδύναμο πρόγραμμα γραμμένο σε μια γλώσσα Τ. Η γλώσσα Α ονομάζεται αρχική γλώσσα και η Τ τελική γλώσσα. Συνήθως η αρχική είναι μία γλώσσα υψηλού επιπέδου και η τελική μία γλώσσα μηχανής κάποιου συγκεκριμένου επεξεργαστή [2]. Όταν κατασκευάζουμε ένα μεταγλωττιστή, στην ουσία ορίζουμε σημασιολογικά την πηγαία γλώσσα προγραμματισμού (τη γλώσσα Α δηλαδή του μεταγλωττιστή) και τις διαδικασίες μετατροπής των συντακτικών μορφωμάτων της πηγαίας γλώσσας σε διαδικασίες που θα πρέπει να εκτελέσει η υποθετική μηχανή του συστήματος εκτέλεσης. Η μηχανή μπορεί να είναι πραγματική (π.χ. ο επεξεργαστής MIPS) ή εικονική (π.χ. JavaVM, SPIM). Η γλώσσα με την οποία θα επιλέξουμε να κατασκευάσουμε τον μεταγλωττιστή είναι συνήθως ανεξάρτητη από την αρχική και την τελική γλώσσα.

Η διαδικασία με την οποία προκύπτει η τελική γλώσσα ονομάζεται μεταγλώττιση και περιλαμβάνει πολλά στάδια που εξαρτώνται συνήθως από το είδος του μεταγλωττιστή και τον τελικό στόχο του. Μπορεί να χωριστεί σε δύο φάσεις, τη φάση της ανάλυσης και τη φάση της σύνθεσης. Η ανάλυση περιλαμβάνει τα παρακάτω στάδια με τη σειρά που εμφανίζονται:

- Λεκτική Ανάλυση
- Συντακτική Ανάλυση
- Σημασιολογική Ανάλυση

Ενώ η σύνθεση περιλαμβάνει τα ακόλουθα τρία στάδια σύνθεσης με τη σειρά που εμφανίζονται:

- Γεννήτορας Ενδιάμεσου Κώδικα
- Βελτιστοποιητές Υψηλού και Μέσου Επιπέδου
- Γεννήτορας Τελικού Κώδικα

Πολύ κρίσιμο στάδιο της δεύτερης φάσης είναι αυτό της γέννησης του ενδιάμεσου κώδικα καθώς επιτρέπει την παραγωγή τελικού κώδικα για διαφορετικές μηχανές (στοχευόμενες αρχιτεκτονικές) χωρίς την επανάληψη των τριών σταδίων της φάσης

της ανάλυσης. Το στάδιο μεταγλώττισης από την πηγαία γλώσσα μέχρι την ενδιάμεση αναπαράσταση (IR) του κώδικα ονομάζεται frontend και από την IR μέχρι την τελική γλώσσα ονομάζεται backend. Ουσιαστικά λοιπόν η ενδιάμεση αναπαράσταση αποτελεί την «κοινή γλώσσα επικοινωνίας» των διάφορων μεταγλωττιστών, καθώς ανεξάρτητα της γλώσσας που είναι γραμμένο το πηγαίο πρόγραμμα, μπορούμε με το κατάλληλο backend κάθε φορά και την ενδιάμεση αναπαράσταση του πηγαίου προγράμματος (ανεξάρτητα από ποια γλώσσα προγραμματισμού προέκυψε) να παράγουμε τον τελικό κώδικα για το στόχο μας [4-9]. Υπάρχουν τρία βασικά είδη ενδιάμεσης αναπαράστασης:

- Συντακτικά δέντρα (Syntax Trees)
- Συμβολισμός Postfix (Postfix Notation)
- Κώδικας Τριών Διευθύνσεων (TAC) ή Τετράδων (Quadruples)

Η ενδιάμεση αναπαράσταση πρέπει να είναι πλήρης και ορθογώνια, ώστε να βοηθά στην παραγωγή τελικού κώδικα. Πρέπει να δοθεί όμως προσοχή, γιατί αν δημιουργηθεί μια ενδιάμεση αναπαράσταση πολύ απλή, θα αναγκαστεί να παράγει ο μεταγλωττιστής υπερμεγέθη ενδιάμεσο κώδικα [3]. Από τα τρία παραπάνω είδη, αυτό που θα μας απασχολήσει σε αυτή την εργασία είναι ο κώδικας τριών διευθύνσεων και πως αυτός μπορεί να πάρει μία άλλη επεκτεταμένη μορφή έτσι ώστε ο ενδιάμεσος κώδικας να είναι μικρής έκτασης με ισχυρή σημασιολογική δύναμη [2]. Ο κώδικας τριών διευθύνσεων είναι μία ακολουθία προτάσεων με την ακόλουθη γενική μορφή:

x = y operator z;

όπου τα x, y και z είναι ονόματα, σταθερές ή προσωρινά ονόματα του μεταγλωττιστή. Το operator μπορεί να είναι οποιοσδήποτε αριθμητικός ή λογικός τελεστής. Σημειώστε ότι δεν επιτρέπονται σύνθετες αριθμητικές εκφράσεις του τύπου x + y * z, καθώς μόνο ένας τελεστής μπορεί να υπάρχει στα δεξιά της κάθε πρότασης. Έτσι η παραπάνω αριθμητική πράξη μπορεί να μεταφραστεί σε TAC όπως παρακάτω:

t_1 = y * z;
t_2 = x + t_1;

όπου t_1 και t_2 προσωρινά ονόματα μεταγλωττιστή. Αυτή η απλοποίηση των πολύπλοκων αριθμητικών εκφράσεων καθώς και των εμφωλευμένων δομών ελέγχου ροής, καθιστούν το TAC ελκυστικό για την παραγωγή και βελτιστοποίηση του τελικού κώδικα μιας μηχανής. Ο λόγος που ονομάζεται κώδικας τριών διευθύνσεων οφείλεται στο γεγονός ότι κάθε πρόταση του περιέχει τρεις διευθύνσεις, δύο για τους όρους της πράξης και μία για το αποτέλεσμα αυτής. Αυτό σε αντίθεση με τη γλώσσα ενδιάμεσης αναπαράστασης NAC (N-address code), που σχεδίασε ο Νικόλαος Καββαδίας [1] και θα δούμε στη συνέχεια, όπου σε κάθε πρόταση το πλήθος των διευθύνσεων των όρων μιας πράξης αλλά και των αποτελεσμάτων αυτής, να μπορεί να είναι απεριόριστο.

Η επιλογή του πλήθους των επιτρεπόμενων τελεστών στο σχεδιασμό μίας ενδιάμεσης μορφής είναι πολύ σημαντική. Το σύνολο αυτό πρέπει να είναι σαφώς αρκετά μεγάλο ώστε να καλύπτει όλες τις πιθανές πράξεις της πηγαίας γλώσσας. Όμως είναι γεγονός ότι ένα μικρότερο σύνολο τελεστών είναι πιο εύκολο να υλοποιηθεί σε μία νέα μηχανή στόχο. Ωστόσο ένα περιορισμένο σύνολο εντολών μπορεί να αναγκάσει το frontend να παράγει μεγάλες ακολουθίες προτάσεων για μερικές πράξεις της πηγαίας γλώσσας. Ο γεννήτορας κώδικα και ο βελτιστοποιητής θα πρέπει να δουλέψουν σκληρά (περισσότερος χρόνος) για τη παραγωγή καλού κώδικα για τη μηχανή στόχο [2].

Για να μπορέσουμε λοιπόν σε αυτή την εργασία να δοκιμάσουμε τη γλώσσα NAC και να μελετήσουμε την ενδιάμεση μορφή της, υλοποιήσαμε ένα διερμηνευτικό προσομοιωτή (Interpretive Simulator) με το όνομα NacVM. Πρόκειται στην ουσία για μία εικονική μηχανή (Virtual Machine) γραμμένη σε ANSI C και ο τρόπος λειτουργίας και κατασκευής της θα μας απασχολήσουν στα επόμενα κεφάλαια αυτής της εργασίας.

Η εικονική μηχανή (VM) είναι ένα λογισμικό που υλοποιεί μία εν δυνάμει φυσική μηχανή (π.χ. επεξεργαστή) και εκτελεί προγράμματα σαν να ήταν αυτή. Οι εικονικές μηχανές χωρίζονται σε δύο κύριες κατηγορίες με βάση τη χρήση τους και το βαθμό της προσομοίωσης μίας οποιασδήποτε πραγματικής μηχανής. Εκείνες που έχουν σχεδιαστεί για να υποστηρίζουν την εκτέλεση ενός απλού προγράμματος, ονομάζονται εικονικές μηχανές διαδικασίας (π.χ. SPIM), σε αντίθεση με τις μηχανές εικονικού συστήματος (π.χ. JavaVM). Ένα ουσιώδες χαρακτηριστικό της εικονικής

μηχανής είναι ότι το λογισμικό που εκτελείται εντός αυτής περιορίζεται στους πόρους και ότι οι αφαιρέσεις που παρέχει δεν μπορούν να ξεφύγουν από τον εικονικό κόσμο της [10].

2. Η Γραμμική Ενδιάμεση Αναπαράσταση NAC

2.1 Η Δομή ενός NAC Προγράμματος

Ο κύριος σκοπός της χρήσης της NAC είναι η εκτελέσιμη/διερμηνευόμενη ενδιάμεση αναπαράσταση για εργαλεία μεταγλώττισης (μεταγλωττιστές, υψηλού επιπέδου εργαλεία σύνθεσης, κ.α.). Οι προτάσεις της NAC μπορεί να είναι ετικέτες, εντολές n – διευθύνσεων ή κλήσεις διαδικασιών.

Ένα πρόγραμμα NAC αποτελείται από ένα απλό αρχείο πηγαίου κώδικα το οποίο περιέχει σχόλια, δηλώσεις και αρχικοποιήσεις καθολικών ή τοπικών μεταβλητών και μία ή περισσότερες διαδικασίες. Τα σχόλια μπορεί να βρίσκονται οπουδήποτε στο αρχείο αρκεί να ξεκινούν πάντα με δύο καθέτους «//». Η πρώτη διαδικασία του αρχείου αποτελεί τη γονική διαδικασία του προγράμματος και είναι εκείνη που δέχεται ορίσματα εισόδου του προγράμματος και παράγει τα δεδομένα εξόδου. Κάθε διαδικασία αποτελείται από τα παρακάτω:

i. Τη λέξη κλειδί procedure
ii. Το όνομα της διαδικασίας
iii. Μία διατεταγμένη λίστα από ορίσματα εισόδου, διαχωρισμένα με κόμμα
iv. Μία διατεταγμένη λίστα από ορίσματα εξόδου, διαχωρισμένα με κόμμα
v. Δηλώσεις και αρχικοποιήσεις τοπικών μεταβλητών
vi. Ένα πλήθος από προτάσεις NAC

Προσοχή! Δεν μπορείτε να βάλετε δύο ή περισσότερες διαδικασίες μέσα σε ένα αρχείο, αν αυτές δεν καλούνται κάποια στιγμή από τη γονική (πρώτη στο αρχείο) διαδικασία, καθώς έτσι δεν πρόκειται να εκτελεστούν ποτέ. Ένα τυπικό πρόγραμμα NAC δομείται όπως φαίνεται στην Εικόνα 1.

```
// comments

<Global variable declarations>

procedure <name-1> ( <input arguments>, <output arguments> )
{
       <Local variable declarations>
       <NAC labels, instructions and procedure calls>
}
...
procedure <name-n> ( <input arguments>, <output arguments> )
{
       <Local variable declarations>
       <NAC labels, instructions and procedure calls>
}
```

Εικόνα 1. Η δομή ενός τυπικού προγράμματος NAC

2.2 Ανάλυση των Στοιχείων της NAC

Στην προηγούμενη ενότητα παρουσιάσαμε γενικά τη δομή ενός προγράμματος NAC. Σε αυτή εδώ την ενότητα θα δείξουμε και θα περιγράψουμε αναλυτικά τους όρους και τις έννοιες που αναφέρθηκαν παραπάνω.

2.2.1 Προτάσεις NAC

- **Ετικέτα**

Μπορεί να είναι οποιοδήποτε αλφαριθμητικό ακολουθούμενο από το σύμβολο ":" (άνω και κάτω τελεία). Κάθε ετικέτα θα πρέπει να βρίσκεται μόνη της σε μία γραμμή στο πηγαίο αρχείο του κώδικα NAC. Αποτελεί στην ουσία σημείο-«άγκυρα» μέσα στον κώδικα, καθώς επιτρέπει τη μετάβαση της ροής του προγράμματος στο σημείο αυτό. Όπως γίνεται κατανοητό απαγορεύεται η χρήση του ίδιου ονόματος για τη δήλωση μίας ετικέτας μέσα σε μία διαδικασία, καθώς οι εντολές άλματος *jmpzz* δεν θα μπορούν να επιλέξουν ανάμεσα τους. Ακολουθεί ένα παράδειγμα, όπου S_1 και S_2 ετικέτες:

```
S_1:
      ix <= ldc 0;
      S_2 <= jmpun;
S_2:
     temp <= setlt ix, inp;
     S_1, S_EXIT <= jmpeq temp, 1;
```

Παρατήρηση: Τα ονόματα των ετικετών χρησιμοποιούνται σαν ορίσματα εξόδου των εντολών που ανήκουν στην οικογένεια *jmpzz*, όπως φαίνεται και στο παράδειγμα

επάνω. Όλες τις διαθέσιμες εντολές της έκδοσης αυτής του NacVM θα τις παρουσιάσουμε αναλυτικά σε επόμενη ενότητα αυτού του κεφαλαίου.

- **Εντολή n–διευθύνσεων**

Στην πραγματικότητα είναι μία προδιαγραφή αντιστοίχησης από ένα σύνολο n διατεταγμένων εισόδων σε ένα σύνολο m διατεταγμένων εξόδων. Με την εντολή εκτελείται η αντίστοιχη «πράξη» με δεδομένα εισόδου τις n μεταβλητές που βρίσκονται δεξιά της εντολής και επιστρέφει τιμή στις m μεταβλητές που βρίσκονται αριστερά της. Μία τέτοια εντολή συντάσσεται όπως παρακάτω:

$outp_1$, …, $outp_m$ <= operation inp_1, …, inp_n ;

όπου,

- *operation*, αναπαριστά εντολή επιπέδου IR
- $outp_1, \ldots, outp_m$, είναι οι m έξοδοι της εντολής
- $inp_1, \ldots, inp_n$, είναι οι n είσοδοι της εντολής

Η συγκεκριμένη έκδοση του NacVM που παρουσιάζουμε περιέχει εντολές με μέγιστο πλήθος εξόδων m = 2 και μέγιστο πλήθος εισόδων n = 4. Το πλήθος των εντολών (operation) που υποστηρίζει είναι 47, συν μία ακόμα η οποία συντάσσεται εντελώς διαφορετικά από τις υπόλοιπες και ονομάζεται *print*. Όμως οι προδιαγραφές υλοποίησης του, επιτρέπουν τον ορισμό νέων εντολών με απεριόριστο αριθμό μεταβλητών εισόδου και εξόδου, καθώς η NAC είναι επεκτάσιμη γλώσσα. Αυτό είναι ένα χαρακτηριστικό που κάνει τη NAC πολύ ισχυρή και αποδοτική γλώσσα ενδιάμεσης αναπαράστασης. Τα παραδείγματα που ακολουθούν στα επόμενα κεφάλαια θα βοηθήσουν στην κατανόηση των δυνατοτήτων της.

- **Κλήση διαδικασίας**

Παρόμοια με μία εντολή, έτσι και μία κλήση διαδικασίας, η οποία αποτελεί μία μη ατομική λειτουργία, συντάσσεται όπως παρακάτω:

(outp$_1$, …, outp$_m$) <= procedure-name (inp$_1$, …, inp$_n$);

όπου,

- *procedure-name*, είναι το όνομα της διαδικασίας που καλείται

Προσοχή! Η διαφορά από την εντολή είναι ότι στην κλήση, οι είσοδοι και οι έξοδοι βρίσκονται μέσα σε παρενθέσεις. Αν στον πηγαίο σας κώδικα, μία διαδικασία που έχετε υλοποιήσει έχει το ίδιο όνομα με μία εντολή του NacVM και ξεχάσετε να βάλετε παρενθέσεις, τότε τα αποτελέσματα θα είναι πραγματικά απρόσμενα.

Λεπτομέρειες για το πως προγραμματίζουμε με αυτά τα τρία είδη προτάσεων της NAC και τι πρέπει να προσέχουμε θα το δούμε στη συνέχεια με αρκετά παραδείγματα.

2.2.2 Δηλώσεις καθολικών και τοπικών μεταβλητών

Αφού παρουσιάσαμε τις προτάσεις που υποστηρίζει ο NacVM, τώρα θα πρέπει να δούμε τους τύπους και τα είδη των μεταβλητών που αυτός υποστηρίζει. Πριν όμως προχωρήσουμε θα πρέπει να πούμε δύο λόγια για τις μεταβλητές, οι οποίες ανεξάρτητα από τον τύπο (s32) και το είδος (localvar, globalvar, in, out) που ανήκουν, αντιμετωπίζονται στη NAC σαν βαθμωτοί μονοδιάστατοι πίνακες. Όταν δεν αρχικοποιούνται στις δηλώσεις τους, ο NacVM τις αρχικοποιεί με την τιμή μηδέν (0). Όταν αρχικοποιούνται από τον προγραμματιστή, τότε οι τιμές αρχικοποίησης τοποθετούνται μέσα σε άγκιστρα και διαχωρίζονται με κόμμα, π.χ. temp[2]={44,-8}. Το μέγεθος μίας μεταβλητής (μονοδιάστατος πίνακας) μπορεί να κυμαίνεται από 1 έως 2147483647 και δηλώνεται με τον αντίστοιχο αριθμό μέσα σε αγκύλες μετά το όνομα της, π.χ. var[24]. Δηλαδή η var είναι ένας μονοδιάστατος πίνακας 24 θέσεων, από τη θέση 0 έως 23. Το χαρακτηριστικό αυτό αποτελεί άλλο ένα δυνατό σημείο της NAC. Η πρώτη θέση της μεταβλητής είναι πάντα η μηδέν (0).

- **Τοπικές μεταβλητές**

Η δεσμευμένη λέξη *localvar* χρησιμοποιείται μέσα στο σώμα κάθε διαδικασίας για να δηλώσει ότι οι μεταβλητές που ακολουθούν “είναι ορατές” μόνο στη συγκεκριμένη διαδικασία. Είναι υποχρεωτική η χρήση της όταν θέλουμε να δηλώσουμε τοπικές μεταβλητές. Παράδειγμα

```
localvar s32 ctr, tmp[2]={24,-8899};
```

όπου ctr και tmp μεταβλητές.

- **Καθολικές μεταβλητές**

Η δεσμευμένη λέξη *globalvar* χρησιμοποιείται έξω από τα σώματα των διαδικασιών για να δηλώσει ότι οι μεταβλητές που ακολουθούν "είναι ορατές" σε όλες τις διαδικασίες που υπάρχουν στο αρχείο του κώδικα . Είναι υποχρεωτική η χρήση της όταν θέλουμε να δηλώσουμε μεταβλητές που να έχουν καθολική εμβέλεια στο πηγαίο αρχείο κώδικα της NAC. Οι καθολικές μεταβλητές θα πρέπει να δηλώνονται στην αρχή του αρχείου. Παράδειγμα

```
globalvar s32 tmp1, tmp2;
```

όπου tmp1 και tmp2 μεταβλητές.

- **Ορίσματα εισόδου**

Η δεσμευμένη λέξη *in* χρησιμοποιείται για να δηλώσει ότι η συγκεκριμένη διαδικασία δέχεται σαν όρισμα μία τιμή (παράμετρος εισόδου), όταν πρόκειται να κληθεί. Παράδειγμα

```
procedure cmul19 (in s32 x, out s32 outp)
{
  localvar s32 t0, t1, t2, t3;

S_1:
  t0 <= mov x;
  t1 <= shl t0, 4;
  t2 <= shl t0, 1;
  t3 <= add t1, t2;
  outp <= add t3, t0;
}
```

όπου η μεταβλητή *x* αποτελεί όρισμα εισόδου.

- **Ορίσματα εξόδου**

Αντίστοιχα η out χρησιμοποιείται για να δηλώσει ότι η συγκεκριμένη διαδικασία επιστρέφει μία τιμή (παράμετρος εξόδου), στο σημείο που κλήθηκε. Η σειρά με την οποία θα εμφανίζονται στη δήλωση της διαδικασίας δεν είναι περιοριστική συντακτικά, αλλά είναι προτιμότερο να δηλώνονται πρώτα οι παράμετροι εισόδου και έπειτα οι εξόδου. Στο παραπάνω παράδειγμα η μεταβλητή *outp* δηλώνεται ως τέτοιο όρισμα.

Σημαντικό: Το πλήθος των ορισμάτων εισόδου και εξόδου των διαδικασιών που δημιουργούμε μπορεί να είναι απεριόριστο, είναι υποχρεωτικό όμως να έχει

τουλάχιστον ένα όρισμα εισόδου και ένα όρισμα εξόδου, ας μην υπάρχει η ανάγκη χρήση τους. Αυτό τον περιορισμό μας τον επιβάλλει ο NacVM και όχι η ίδια η γλώσσα NAC. Επίσης όσον αφορά τα ορίσματα της κλήσης διαδικασίας θα πρέπει να βρίσκονται σε αντιστοιχία ένα-προς-ένα με αυτά της διαδικασίας. Και τα δύο είδη (εισόδου-εξόδου) θεωρούνται τοπικές μεταβλητές και μπορούν να χρησιμοποιηθούν στην αντίστοιχη διαδικασία σαν τις *localvar*. Έστω η παρακάτω διαδικασία

```
procedure cmul2 (in s32 x, in s32 y, out s32 outx, out s32 outy)
{
  localvar s32 t0,t1;

S_1:
  t0 <= mov x;
  t1 <= mov y;
  outx <= shl t0, 1;
  outy <= rotl t1, 3;
}
```

την οποία θέλουμε να την καλέσουμε σε κάποιο σημείο του κώδικα μας, τότε η κλήση διαδικασίας θα έχει κάπως έτσι:

```
(out1, out2) <= cmul2 (in1, in2);
```

και επειδή ισχύει το *pass by value* στη NAC, θα έχουμε x = in1 και y = in2. Ενώ όταν η διαδικασία επιστρέψει τα δεδομένα θα έχουμε out1 = outx και out2 = outy. Για το λόγο αυτό στο iii. και iv. της Ενότητας 3.1 κάναμε λόγο για διατεταγμένη λίστα ορισμάτων εισόδου και εξόδου. Επειδή το μέγεθος των μεταβλητών μπορεί να είναι μεγαλύτερο από ένα, θα πρέπει να υπάρχει και αντιστοιχία μεταξύ του μεγέθους τους. Δηλαδή αν η x έχει μέγεθος τρία τότε και η in1 θα πρέπει να έχει το ίδιο μέγεθος. Το ίδιο ισχύει και για τα ορίσματα εξόδου. Από ότι καταλάβατε η NAC επιτρέπει τη κλήση διαδικασίας με ορίσματα (μεταβλητές) που έχουν μέγεθος μεγαλύτερο από ένα (χαρακτηριστικό που ενισχύει τη δύναμη της γλώσσας). Προσοχή όμως να μη ξεχνάτε την αντιστοιχία μεγέθους των ορισμάτων στις κλήσεις των διαδικασιών.

Αφού δείξαμε τα είδη των μεταβλητών, θα παρουσιάσουμε τους δυνατούς τύπους των μεταβλητών. Δυστυχώς σε αυτή την έκδοση της NAC, υποστηρίζεται μόνο ο τύπος s32, δηλαδή μεταβλητές με προσημασμένες ακέραιες τιμές μεγέθους 32 bit. Το εύρος των τιμών τους δηλαδή (όχι το μέγεθος της μεταβλητής!) θα μπορεί να κυμαίνεται από $-(2^{31})$ μέχρι $2^{31}-1$. Σε διαφορετική περίπτωση θα παράγεται κωδικός λάθους.

Ακολουθούν παραδείγματα δήλωσης και αρχικοποίησης μεταβλητών.

- ΣΩΣΤΟ
  - `localvar s32 var;`
    var[0]=0;
  - `localvar s32 tmp1[5];`
    Ο NacVM αρχικοποιεί τη μεταβλητή με μηδέν, δηλαδή οι θέσεις από μηδέν μέχρι τέσσερα της tmp1 παίρνουν την τιμή 0. Δηλαδή tmp1[0] = 0,…, tmp2[4] = 0.
  - `globalvar s32 test[3]={245,-14438776,0};`
    Με τον τρόπο αυτό η test θα έχει ως εξής test[0] = 245, test[1] = -14438776 και test[2] = 0. **Προσοχή!** Στην αρχικοποίηση μεταβλητών δεν θα πρέπει να υπάρχει κενό (whitespace) πριν και μετά του «=».
  - `localvar s32 var[1];`
    Σωστό, αλλά είναι πλεονασμός! Το var[1] είναι ισοδύναμο με το var σκέτο, καθώς αν δε δηλωθεί το μέγεθος, η μεταβλητή θεωρείται μεγέθους 1 με τιμή 0.

- ΛΑΘΟΣ
  - `globalvar s32 p[2]={1};`
    Αρχικοποίηση μόνο της πρώτης θέσης. Θα πρέπει είτε να τις αρχικοποιούμε όλες ή καμία.
  - `globalvar s32 t0[1] = {5445};`
    whitespaces πριν και μετά το =
  - `localvar s32 value[1]={};`
  - `localvar s32 variable[0];`

2.3 Οι Εντολές της NAC

Στην ενότητα αυτή θα παρουσιάσουμε αναλυτικά τις εντολές n – διευθύνσεων που υποστηρίζει ο NacVM, ο πηγαίος κώδικας των οποίων βρίσκεται στο Παράρτημα Α της εργασίας αυτής. Επειδή η NAC είναι επεκτάσιμη γλώσσα, ο NacVM επιτρέπει τον ορισμό και τη δημιουργία νέων εντολών n – διευθύνσεων, πέραν των παρακάτω ενσωματωμένων (built-in). Για αυτό το λόγο σκοπεύουμε αργότερα να δείξουμε τον τρόπο με τον οποίο κάποιος μπορεί εύκολα να ορίσει νέες εντολές στον NacVM. Οι περισσότερες εντολές μπορούν να δέχονται ως ορίσματα εισόδου, εκτός από

μεταβλητές και αριθμητικά κυριολεκτικά στοιχεία (literals). Για ποιες εντολές ισχύει αυτό και σε ποιες θέσεις επιτρέπεται θα το δούμε ακολούθως. Για όλες τις παρακάτω που θα δείτε, αλλά και για αυτές που πρόκειται να ορίσετε εσείς οι ίδιοι αν κάποτε το θελήσετε, ισχύει το εξής σημαντικό: Το μέγεθος των μεταβλητών εισόδου και εξόδου μπορεί να είναι μεγαλύτερο από ένα, **όμως** οι εντολές λαμβάνουν και επιστρέφουν τιμές μόνο στην πρώτη θέση αυτών. Παράδειγμα:

```
localvar s32 myVar1[3]={67,90,-13}, myVar2[2]={13,24}, result[4];
result <= add myVar1, myVar2;
```

Ποια θα είναι η τιμή της result μετά την εντολή add;

Πριν	Μετά
result[0]=0	result[0]=80
result[1]=0	result[1]=0
result[2]=0	result[2]=0
result[3]=0	result[3]=0

- **nop**

  ```
  nop;
  ```

  Κάθε ετικέτα για να μπορεί να «υπάρξει» στη στοίβα μηχανής πρέπει στην εμβέλεια της να υπάρχει ένα τουλάχιστον statement. Όταν λοιπόν στο πρόγραμμα μας θέλουμε να κάνουμε έξοδο, κάνουμε jump στη τελευταία ετικέτα της γονικής διαδικασίας, η οποία δε θέλουμε να έχει στην εμβέλεια της κάποια εκτελέσιμη εντολή. Τότε είναι που μας χρειάζεται το nop, για να δηλώσουμε no-operation. Παράδειγμα:

  ```
  …
  S_EXIT:
        nop;
  }
  ```

- **mov**

  ```
  dst1 <= mov src1;
  ```

  Αντιγράφει το περιεχόμενο της μεταβλητής src1 στη dst1. Στη θέση της src1 μπορεί να υπάρχει ένας προσημασμένος ακέραιος αριθμός.

- **ldc**

  ```
  dst1 <= ldc src1;
  ```

  Αντιγράφει το περιεχόμενο της μεταβλητής src1 στη dst1. Στη θέση της src1 μπορεί να υπάρχει ένας προσημασμένος ακέραιος αριθμός.

- **jmpun (Unconditional jump)**

  ```
  S_dst1 <= jmpun;
  ```

  Μεταφέρει τη ροή του προγράμματος στην ετικέτα S_dst1 που βρίσκεται στην ίδια διαδικασία που βρίσκεται και η εντολή. Είχαμε αναφέρει σε προηγούμενη ενότητα ότι δεν θα πρέπει να υπάρχουν ετικέτες με το ίδιο όνομα μέσα σε μία διαδικασία και ότι τα ορίσματα εξόδου των εντολών της οικογένειας jmpzz είναι ετικέτες, όπως θα δείτε και παρακάτω.

- **jmpeq (Conditional jump)**

  ```
  S_TRGT, S_TRGF <= jmpeq src1, src2; //jump if equal
  ```

  Αν η τιμή της src1 είναι ίση με αυτή της src2, τότε η ροή μεταφέρεται στην ετικέτα S_TRGT, διαφορετικά στη S_TRGF. Γενικά και για τα παρακάτω άλματα υπό συνθήκη (conditional jumps), ισχύει το εξής: αν η συνθήκη είναι αληθής τότε πήγαινε στο πρώτο όρισμα εξόδου (ετικέτα) της εντολής, διαφορετικά στο δεύτερο. Στη θέση της src2 μπορεί να υπάρχει ένας προσημασμένος ακέραιος αριθμός. Και αυτό ισχύει για όλα τα άλματα υπό συνθήκη που ακολουθούν.

- **jmpne (Conditional jump)**

  ```
  S_TRGT, S_TRGF <= jmpne src1, src2; //jump if not equal
  ```

  Αν η τιμή της src1 δεν είναι ίση με αυτή της src2, τότε η ροή μεταφέρεται στην ετικέτα S_TRGT, διαφορετικά στη S_TRGF.

- **jmplt (Conditional jump)**

  ```
  S_TRGT, S_TRGF <= jmplt src1, src2; //jump if less than
  ```

  Αν η τιμή της src1 είναι μικρότερη από της src2, τότε η ροή μεταφέρεται στην ετικέτα S_TRGT, διαφορετικά στη S_TRGF.

- **jmple (Conditional jump)**

  ```
  S_TRGT, S_TRGF <= jmple src1, src2; //jump if less or equal
  ```

  Αν η τιμή της src1 είναι μικρότερη ή ίση από της src2, τότε η ροή μεταφέρεται στην ετικέτα S_TRGT, διαφορετικά στη S_TRGF.

- **jmpgt (Conditional jump)**

  `S_TRGT, S_TRGF <= jmpgt src1, src2; //jump if greater than`

  Αν η τιμή της src1 είναι μεγαλύτερη από της src2, τότε η ροή μεταφέρεται στην ετικέτα S_TRGT, διαφορετικά στη S_TRGF.

- **jmpge (Conditional jump)**

  `S_TRGT, S_TRGF <= jmpge src1, src2; //jump if greater or equal`

  Αν η τιμή της src1 είναι μεγαλύτερη ή ίση από της src2, τότε η ροή μεταφέρεται στην ετικέτα S_TRGT, διαφορετικά στη S_TRGF.

- **Binary Bitwise Instructions – Με *παραδείγματα***

  Στη θέση της μεταβλητής src2 μπορεί να υπάρχει ένας προσημασμένος ακέραιος αριθμός. Δηλαδή result <= BBI var1, -889; όπου BBI μία από τις παρακάτω Binary Bitwise Instructions.

  Έστω: `localvar s32 src1[1]={5}, src2[1]={3}, dst1;`

  - **and**

    `dst1 <= and src1, src2;`

    Τότε result = 1

  - **ior (inclusive-or)**

    `dst1 <= ior src1, src2;`

    Τότε result = 7

  - **xor (exclusive-or)**

    `dst1 <= xor src1, src2;`

    Τότε result = 6

  - **nand**

    `dst1 <= nand src1, src2;`

    Τότε result = -2, επειδή src1 και src2 προσημασμένοι ακέραιοι.

  - **nor**

    `dst1 <= nor src1, var2;`

    Τότε result = -8, επειδή src1 και src2 προσημασμένοι ακέραιοι.

  - **xnor**

    `dst1 <= xnor src1, var2;`

    Τότε result = -7, επειδή src1 και src2 προσημασμένοι ακέραιοι.

- **not (Unary bitwise instruction)**

```
dst1 <= not src1;
```

Αντιγράφει το συμπλήρωμα ως προς 1 (1's complement) της τιμής της src1 στη dst1. Στη θέση της src1 μπορεί να υπάρχει ένας προσημασμένος ακέραιος αριθμός.

- **add (Binary arithmetic instruction)**

```
dst1 <= add src1, src2;
```

Προσθέτει τις τιμές των src1 και scr2 ως προς συμπλήρωμα του 2 και το αποτέλεσμα το αποθηκεύει στη dst1. Στις θέσεις των src1 και src2 μπορεί να υπάρχουν προσημασμένοι ακέραιοι αριθμοί. Ακολουθούν έγκυρα παραδείγματα:

```
dst1 <= add src1, -787;
dst1 <= add 567, src2;
dst1 <= add -89, 10;
```

- **sub (Binary arithmetic instruction)**

```
dst1 <= sub src1, src2;
```

Αφαιρεί από την τιμή της src1 την τιμή της src2. Το αποτέλεσμα το αποθηκεύει στη dst1. Όπως και με την add, στις θέσεις των src1 και src2 μπορεί να υπάρχουν προσημασμένοι ακέραιοι αριθμοί.

- **neg (Unary arithmetic instruction)**

```
dst1 <= neg src1;
```

Αντιγράφει την αντίθετη έκδοση της τιμής της src1 στη dst1. Στη θέση της src1 μπορεί να υπάρχει ένας προσημασμένος ακέραιος αριθμός.

- **muxeq (Quaternary multiplexing instruction)**

```
dst1 <= muxeq src1, src2, src3, src4;
```

Σε αυτή εδώ την εντολή συγκρίνονται οι δύο πρώτες μεταβλητές εισόδου, δηλαδή οι src1 και scr2. Αν οι τιμές τους είναι ίσες τότε στη dst1 αντιγράφεται η τιμή της src3, διαφορετικά αντιγράφεται της src4. Στις θέσεις των src2 και src4 μπορεί να υπάρχουν προσημασμένοι ακέραιοι αριθμοί. Πιθανές μορφές της muxeq:

```
dst1 <= muxeq src1, 63, src3, src4;
dst1 <= muxeq src1, -225, src3, 5;
dst1 <= muxeq src1, src2, src3, 123;
dst1 <= muxeq src1, src2, src3, src4;
```

Το παραπάνω ισχύει για όλες τις εντολές της οικογένειας muxzz.

- **muxne (Quaternary multiplexing instruction)**

  `dst1 <= muxne src1, src2, src3, src4;`

  Αν οι τιμές των src1 και src2 είναι άνισες τότε στη dst1 αντιγράφεται η τιμή της src3, αλλιώς αντιγράφεται αυτή της src4.

- **muxlt (Quaternary multiplexing instruction)**

  `dst1 <= muxlt src1, src2, src3, src4;`

  Αν η τιμή της src1 είναι μικρότερη από αυτή της src2 τότε στη dst1 αντιγράφεται η τιμή της src3, αλλιώς αντιγράφεται αυτή της src4.

- **muxle (Quaternary multiplexing instruction)**

  `dst1 <= muxle src1, src2, src3, src4;`

  Αν η τιμή της src1 είναι μικρότερη ή ίση από αυτή της src2 τότε στη dst1 αντιγράφεται η τιμή της src3, αλλιώς αντιγράφεται αυτή της src4.

- **muxgt (Quaternary multiplexing instruction)**

  `dst1 <= muxlt src1, src2, src3, src4;`

  Αν η τιμή της src1 είναι μεγαλύτερη από αυτή της src2 τότε στη dst1 αντιγράφεται η τιμή της src3, αλλιώς αντιγράφεται αυτή της src4.

- **muxge (Quaternary multiplexing instruction)**

  `dst1 <= muxle src1, src2, src3, src4;`

  Αν η τιμή της src1 είναι μεγαλύτερη ή ίση από αυτή της src2 τότε στη dst1 αντιγράφεται η τιμή της src3, αλλιώς αντιγράφεται αυτή της src4.

- **seteq (Set on comparison instruction)**

  `dst1 <= seteq src1, src2;`

  Αν οι τιμές των src1 και src2 είναι ίσες τότε η τιμή της dst1 γίνεται ίση με ένα (μη μηδενική). Διαφορετικά γίνεται ίση με μηδέν. Στη θέση της src2 μπορεί

να είναι ένας προσημασμένος ακέραιος αριθμός (αριθμητικό κυριολεκτικό). Αυτό ισχύει για όλες τις εντολές της οικογένειας setzz που ακολουθούν.

- **setne (Set on comparison instruction)**

  ```
  dst1 <= setne src1, src2;
  ```

  Αν οι τιμές των src1 και src2 είναι άνισες τότε η τιμή της dst1 γίνεται ίση με ένα (μη μηδενική). Διαφορετικά γίνεται ίση με μηδέν.

- **setlt (Set on comparison instruction)**

  ```
  dst1 <= setlt src1, src2;
  ```

  Αν η τιμή της src1 είναι μικρότερη της src2 τότε η τιμή της dst1 γίνεται ίση με ένα (μη μηδενική). Διαφορετικά γίνεται ίση με μηδέν.

- **setle (Set on comparison instruction)**

  ```
  dst1 <= setle src1, src2;
  ```

  Αν η τιμή της src1 είναι μικρότερη ή ίση της src2 τότε η τιμή της dst1 γίνεται ίση με ένα (μη μηδενική). Διαφορετικά γίνεται ίση με μηδέν.

- **setgt (Set on comparison instruction)**

  ```
  dst1 <= setgt src1, src2;
  ```

  Αν η τιμή της src1 είναι μεγαλύτερη της src2 τότε η τιμή της dst1 γίνεται ίση με ένα (μη μηδενική). Διαφορετικά γίνεται ίση με μηδέν.

- **setge (Set on comparison instruction)**

  ```
  dst1 <= setgt src1, src2;
  ```

  Αν η τιμή της src1 είναι μεγαλύτερη ή ίση της src2 τότε η τιμή της dst1 γίνεται ίση με ένα (μη μηδενική). Διαφορετικά γίνεται ίση με μηδέν.

- **abs (Complex unary arithmetic instruction)**

  ```
  dst1 <= abs src1;
  ```

  Αντιγράφει την απόλυτη τιμή της src1 στη dst1. Στη θέση της src1 ΔΕΝ μπορεί να υπάρχει προσημασμένος ακέραιος αριθμός (σταθερά).

- **max (Complex binary arithmetic instruction)**

  ```
  dst1 <= max src1, src2;
  ```

Εκχωρεί στη dst1 τη μεγαλύτερη από τις τιμές των src1 και src2. Στη θέση της src2 μπορεί να υπάρχει προσημασμένος ακέραιος αριθμός.

- **min (Complex binary arithmetic instruction)**

```
dst1 <= min src1, src2;
```

Εκχωρεί στη dst1 τη μικρότερη από τις τιμές των src1 και src2. Στη θέση της src2 μπορεί να υπάρχει προσημασμένος ακέραιος αριθμός.

- **shl (Shift instruction)**

```
dst1 <= shl src1, src2;
```

Στη dst1 αποθηκεύεται το αποτέλεσμα της αριστερής ολίσθησης της τιμής που έχει η src1. Το πλήθος των αριστερών ολισθήσεων εκφράζεται στην τιμή της src2. Στη θέση της src2 μπορεί να υπάρχει θετικός ακέραιος αριθμός. Παράδειγμα

```
localvar s32 src1[1]={10}, dst1;
dst1 <= shl src1, 2;
```

Πριν: 00001010
Μετά: 00101000

Άρα μετά τη παραπάνω εντολή ολίσθησης η τιμή της dst1 θα είναι 40.

- **shr (Shift instruction)**

```
dst1 <= shr src1, src2;
```

Στη dst1 αποθηκεύεται το αποτέλεσμα της δεξιάς ολίσθησης της τιμής που έχει η src1. Το πλήθος των δεξιών ολισθήσεων εκφράζεται στην τιμή της src2. Στη θέση της src2 μπορεί να υπάρχει θετικός ακέραιος αριθμός. Παράδειγμα

```
localvar s32 src1[1]={10}, dst1;
dst1 <= shr src1, 2;
```

Πριν: 00001010
Μετά: 00000010

Άρα μετά την παραπάνω εντολή ολίσθησης η τιμή της dst1 θα είναι 2.

- **rotl (Rotate instruction)**

```
dst1 <= rotl src1, src2;
```

Αριστερή περιστροφή της τιμής της src1. Το πλήθος των περιστροφών εκφράζεται από την τιμή της src2. Στη θέση της src2 μπορεί να υπάρχει θετικός ακέραιος αριθμός. Παράδειγμα

```
localvar s32 src1[1]={17},src2[1]={30},dst1;
dst1 <= rotl src1, src2;
```

Πριν: 00000000 00000000 00000000 00010001 = 17
Μετά: 01000000 00000000 00000000 00000100 = 1073741828

Άρα μετά τη παραπάνω εντολή περιστροφής η τιμή της dst1 θα είναι 1.073.741.828

- **rotr (Rotate instruction)**

```
dst1 <= rotr src1, src2;
```

Δεξιά περιστροφή της τιμής της src1. Το πλήθος των περιστροφών εκφράζεται από την τιμή της src2. Στη θέση της src2 μπορεί να υπάρχει θετικός ακέραιος αριθμός. Παράδειγμα

```
localvar s32 src1[1]={17},src2[1]={30},dst1;
dst1 <= rotr src1, src2;
```

Πριν: 00000000 00000000 00000000 00010001 = 17
Μετά: 00000000 00000000 00000000 01000100 = 68

Άρα μετά την παραπάνω εντολή περιστροφής η τιμή της dst1 θα είναι 68.

- **mul (Multiplication instruction)**

```
dst1 <= mul src1, src2;
```

Πολλαπλασιάζει την τιμή της src1 με αυτή της src2 και αποθηκεύει το αποτέλεσμα στη dst1. Στη θέση της src2 μπορεί να υπάρχει προσημασμένος ακέραιος αριθμός.

- **div (Division instruction)**

```
dst1 <= div src1, src2;
```

Διαιρεί τις τιμές των src1 και src2 και αποθηκεύει το πηλίκο στη dst1. Η src2 δεν μπορεί να έχει την τιμή μηδέν. Ο NacVM θα παραγάγει runtime error στην περίπτωση που θα επιχειρηθεί διαίρεση με το μηδέν. Κι εδώ στη θέση της src2 μπορεί να υπάρχει μη μηδενικός προσημασμένος ακέραιος αριθμός.

- **rem (Division instruction)**

```
dst1 <= rem src1, src2;
```

Διαιρεί τις τιμές των src1 και src2 και αποθηκεύει το υπόλοιπο της διαίρεσης στη dst1. Η src2 δεν μπορεί να έχει την τιμή μηδέν. Ο NacVM θα παραγάγει runtime error στην περίπτωση που θα επιχειρηθεί διαίρεση με το μηδέν. Κι εδώ στη θέση της src2 μπορεί να υπάρχει μη μηδενικός προσημασμένος ακέραιος αριθμός.

- **divrem (Combined division-modulus instruction)**

```
dst1, dst2 <= divrem src1, src2;
```

Διαιρεί τις τιμές των src1 και src2 και αποθηκεύει το πηλίκο στη dst1 και το υπόλοιπο στη dst2. Η src2 δεν μπορεί να έχει την τιμή μηδέν. Ο NacVM θα παραγάγει runtime error στην περίπτωση που θα επιχειρηθεί διαίρεση με το μηδέν. Κι εδώ στη θέση της src2 μπορεί να υπάρχει μη μηδενικός προσημασμένος ακέραιος αριθμός. Παράδειγμα,

```
localvar s32 src1[1]={17},src2[1]={3},dst1,dst2;
dst1,dst2 <= divrem src1, src2;
```

Αποτέλεσμα που παράγει ο NacVM μετά την εκτέλεση της divrem.

```
dst1:
Hex:[5] - Dec:[5]

dst2:
Hex:[2] - Dec:[2]
```

- **zxt (Data type/bitwidth conversion instruction)**

```
dst1 <= zxt
src1;
```

Επέκταση μηδενικού της src1 σε μεγαλύτερο εύρος bit, ίσο με της dst1.

- **sxt (Data type/bitwidth conversion instruction)**

```
dst1 <= sxt src1;
```

Επέκταση πρόσημου της src1 σε μεγαλύτερο εύρος bit, ίσο με της dst1.

- **trunc (Data type/bitwidth conversion instruction)**

```
dst1 <= trunc src1;
```

Περικοπή του εύρους της src1 σε μικρότερο εύρος bit, ίσο με της dst1.

Οι παραπάνω τρεις εντολές προϋποθέτουν την ύπαρξη και άλλων τύπων δεδομένων, εκτός του s32 που υποστηρίζει αυτή τη στιγμή ο NacVM. Οπότε θα μας είναι χρήσιμες σε μία νεότερη έκδοση του NacVM που θα υποστηρίζει και άλλους τύπους. Στη συνέχεια θα παρουσιάσουμε τρεις εντολές πολύ σημαντικές για τον προγραμματισμό εφαρμογών σε NAC.

- **load (Load variable from array)**

  ```
  dst1 <= load src1, src2;
  ```

  Μεταφορτώνει το περιεχόμενο που βρίσκεται στη θέση src2 του πίνακα src1 στη μεταβλητή dst1. Θυμηθείτε ότι η πρώτη θέση ενός πίνακα είναι η μηδέν. Ο dst1 μπορεί να είναι και αυτός πίνακας (δηλαδή μεταβλητή με μέγεθος μεγαλύτερο του ένα), αλλά το αποτέλεσμα της load θα αντιγραφεί στη θέση μηδέν αυτού. Στη θέση της src2 μπορεί να υπάρχει μόνο θετικός ακέραιος αριθμός. Παραδείγματα:

  ```
  1:procedure mytest1(in s32 a, out s32 b)
  2:{
  3:     localvar s32 src1[3]={17,-90,22879},src2, dst1;
  4:
  5:     S:
  6:         dst1 <= load src1, src2;
  7:         print dst1;
  8:}
  ```

  Αποτέλεσμα,

  ```
  dst1:
  Hex:[11] - Dec:[17]
  ```

  Αν στη γραμμή 6 γράψουμε dst1 <= load src1, 1;

  Τότε,

  ```
  dst1:
  Hex:[ffffffa6] - Dec:[-90]
  ```

  Ενώ αν γράψουμε dst1 <= load src1, 3;

  Τότε

  ```
  Error: Array [src1] out of bounds.
  ```

- **store (Store variable to array)**

  ```
  dst1 <= store src1, src2;
  ```

  Αποθηκεύει την τιμή της μεταβλητής src1 στο πίνακα dst1 στη θέση src2. Ο src1 μπορεί να είναι και αυτός πίνακας (δηλαδή μεταβλητή με μέγεθος μεγαλύτερο του ένα), αλλά η store αντιγράφει το περιεχόμενο στη θέση μηδέν αυτού. Στη θέση της src2 μπορεί να υπάρχει μόνο θετικός ακέραιος αριθμός. Παραδείγματα:

  ```
  1:procedure mytest2(in s32 a, out s32 b)
  2:{
  3:     localvar s32 src1[3]={17,-90,22879},src2[1]={2}, dst1[5];
  4:
  5:     S:
  6:         dst1 <= store src1, src2;
  7:         print dst1;
  8:}
  ```

  Αποτέλεσμα:

  ```
  dst1:
  Hex:[0]  - Dec:[0]
  Hex:[0]  - Dec:[0]
  Hex:[11] - Dec:[17]
  Hex:[0]  - Dec:[0]
  Hex:[0]  - Dec:[0]
  ```

  ```
  1:procedure mytest3(in s32 a, out s32 b)
  2:{
  3:     localvar s32 src1[1]={-22879},src2[5]={4,-900,1,87,0}, dst1[5];
  4:
  5:     S:
  6:         dst1 <= store src1, src2;
  7:         print dst1;
  8:}
  ```

  Αποτέλεσμα:

  ```
  dst1:
  Hex:[0]        - Dec:[0]
  Hex:[0]        - Dec:[0]
  Hex:[0]        - Dec:[0]
  Hex:[0]        - Dec:[0]
  Hex:[ffffa6a1] - Dec:[-22879]
  ```

- **print (Print contents of a variable in hexadecimal and decimal form)**

  print variable;

Η variable μπορεί να είναι είτε localvar, globalvar, ή in-out όρισμα διαδικασίας. Ανεξαρτήτως του μεγέθους της variable, η print θα τυπώσει τα περιεχόμενα όλων των θέσεων της σε δεκαεξαδική και σε δεκαδική μορφή. Χρησιμοποιείστε την μόνο σε περιπτώσεις όπου το αρχείο δεδομένων δοκιμής περιλαμβάνει μόνο ένα σενάριο. Λόγος είναι ότι μόνο τότε τα αποτελέσματα που τυπώνει είναι ευανάγνωστα. Το αρχείο δεδομένων δοκιμής παρουσιάζεται στο επόμενο κεφάλαιο.

3. Σχεδίαση Διερμηνευτικού Προσομοιωτή για την NAC

Στο Κεφάλαιο αυτό θα αναλύσουμε και θα επεξηγήσουμε την υλοποίηση του προσομοιωτή NacVM. Θα μιλήσουμε για τους λεκτικούς και γραμματικούς κανόνες που εφαρμόζει, τις θεμελιώσεις δομές (στοίβα μηχανής, πίνακας συμβόλων) που «χτίζει» κατά τη διάρκεια της συντακτικής ανάλυσης του προγράμματος NAC που εισάγουμε, καθώς επίσης και για όλες εκείνες τις βοηθητικές δομές που του επιτρέπουν την έξυπνη και αποτελεσματική προσομοίωση των προγραμμάτων. Τέλος θα δείξουμε τον τρόπο με τον οποίο ο NacVM δέχεται δεδομένα εισόδου και παράγει τα αποτελέσματα. Η γλώσσα προγραμματισμού που χρησιμοποιήσαμε για την ανάπτυξη του είναι η ANSI C ενώ τα εργαλεία ανάπτυξης ήταν ο μεταγλωττιστής gcc, ο λεκτικός αναλυτής flex και ο συντακτικός αναλυτής bison. Όλα αυτά τα συνδυάσαμε κάτω από το ολοκληρωμένο περιβάλλον ανάπτυξης λογισμικού NetBeans IDE 7.0.1. Ο κώδικας των συναρτήσεων που θα αναφέρουμε και θα αναλύουμε σε αυτό το κεφάλαιο, υπάρχει στο Παράρτημα Β της εργασίας αυτής.

3.1 Λεκτικοί Κανόνες

Οι λεκτικοί κανόνες είναι οι πρώτοι που εφαρμόζονται στο πηγαίο πρόγραμμα «διαβάζοντας» τα κατάλληλα tokens, τα οποία τα στέλνουν σαν είσοδο στον συντακτικό αναλυτή με σκοπό την επαλήθευση των γραμματικών κανόνων που ισχύουν για τη γλώσσα NAC.

```
// Name    : nac.l
// Purpose : Lexer for the NAC (n-address code) file parser.
// Author  : Angelos Kanatsos (C) 2011
// Date    : 20-Jul-2011
```

```
%{
#include <stdio.h>
#include <ctype.h>
#include <string.h>
#include <stdlib.h>
#include "nac.tab.h"

#define YY_NO_UNISTD_H 1  /* To avoid the inclusion of <unistd.h>. */

void count();
unsigned long _LINES_ = 0;
int column = 0;
%}

ws      [ \t\f\r]
letter  [a-zA-Z]
integer [1-9][0-9]*
anum    [+-]?[a-zA-Z0-9_][a-zA-Z0-9_]*
A [aA]
B [bB]
C [cC]
D [dD]
E [eE]
F [fF]
G [gG]
H [hH]
I [iI]
J [jJ]
K [kK]
L [lL]
M [mM]
N [nN]
O [oO]
P [pP]
Q [qQ]
R [rR]
S [sS]
T [tT]
U [uU]
V [vV]
W [wW]
X [xX]
Y [yY]
Z [zZ]

%%
"//".*          { count(); }
{ws}            { /* just eat-up whitespace */; }
"\n"            {_LINES_++;column = 0;}
"("             { count(); return(T_LPAREN); }
")"             { count(); return(T_RPAREN); }
"{"             { count(); return(T_LBRACE); }
"}"             { count(); return(T_RBRACE); }
"["             { count(); return(T_LBRACKET); }
"]"             { count(); return(T_RBRACKET); }
","             { count(); return(T_COMMA); }
":"             { count(); return(T_COLON); }
";"             { count(); return(T_SEMI); }
```

```
"<="            { count(); return(T_ASSIGN); }
"="             { count(); return(T_EQUAL); }

{P}{R}{O}{C}{E}{D}{U}{R}{E} { count(); return(T_PROCEDURE); }
{L}{O}{C}{A}{L}{V}{A}{R}    { count(); return(T_LOCALVAR); }
{G}{L}{O}{B}{A}{L}{V}{A}{R} { count();return(T_GLOBALVAR);
                              naclval.StringVal=(char *)strdup(nactext);}
{I}{N}       { count(); return(T_IN); }
{O}{U}{T}    { count(); return(T_OUT); }

{anum}"["{anum}"]""="{"{anum}(","{anum})*"}"   { count(); naclval.StringVal
= (char *)strdup(nactext); return(T_ID_VAR);}
{anum}("["({anum})?"]")?                        { count(); naclval.StringVal
= (char *)strdup(nactext);return(T_ID);}

{anum}?"."{anum}  { count(); return(T_ID); }

.                    { fprintf(stderr, "Error: Unacceptable input character
(0x%x).\n",
                  yytext[0]);
                  exit(1); }
%%

int yywrap()
{
  return(1);
}
void count()
{
  int i;

  for (i=0; yytext[i]!='\0'; i++)
  {
    if (yytext[i] == '\t')
    {
      column += 8 - (column % 8);
    }
    else
    {
      column++;
    }
  }
}
```

3.2 Συντακτικοί κανόνες

Όπως θα δείτε παρακάτω, δίπλα σε κάποιους γραμματικούς κανόνες υπάρχει κάποιος κώδικας που εκτελείται όταν αυτοί επαληθευτούν. Ο κώδικας που εκτελείται βρίσκεται μέσα σε άγκιστρα και συνήθως καλεί τις κατάλληλες συναρτήσεις που «χτίζουν» τις θεμελιώσεις δομές του NacVM με τις πληροφορίες που έχουν κάθε φορά τα τερματικά σύμβολα. Ο συντακτικός αναλυτής ξεκινάει με την κλήση της *nacparse()* που βρίσκεται στη *main()* του προγράμματος του προσομοιωτή.

```
%{
/*
 * Name    : nac.y
 * Purpose : Syntax processing for the NAC (n-address code) formalism.
 * Authors : Nikolaos Kavvadias, Angelos Kanatsos (C) 2011
 * Date    : 20-Jul-2011
 */

  #include <stdio.h>
  #include <stdlib.h>
  #include <string.h>
  #include <stdarg.h>
  #include "stack_mac_nac.h"
  #include "sym_tab_nac.h"

  #define YYDEBUG 1

  extern char *yytext;
  extern char *CUR_PROC_NAME;  //Το όνομα της τρέχουσας διαδικασίας
  extern char *CUR_LABEL_NAME; //Το όνομα της τρέχουσας ετικέτας
  extern char *OPERATION;      //Ο πιο πρόσφατος τελεστής που έχει αναλύσει ο parser

  extern char PCALL;    //Πιθανές τιμές είναι, 0 και 1. Όταν γίνεται 1 σημαίνει
                        //ότι τα περιεχόμενα της στοίβας αφορούν κλήση διαδικασίας

  extern char *GLOBALVAR_TYPE; //Ο τρέχον τύπος των καθολικών μεταβλητών που
                               //αναλύονται

  extern char *PROCEDURE_INOUT_TYPE; //Ο τρέχον τύπος των μεταβλητών εισόδου και
                                     //εξόδου της διαδικασίας που αναλύεται

  extern char *LOCALVAR_TYPE; //Ο τρέχον τύπος των τοπικών μεταβλητών που
                              //αναλύονται

unsigned long  _ERRORS_ = 0;
%}

%union{
    char *StringVal;
}

// Δηλώσεις των τερματικών συμβόλων. Παρατηρούμε ότι το T_ID και το T_ID_VAR αποτελούν
// tokens με τιμές τύπου String κάθε φορά. Καταλαβαίνετε λοιπόν ότι θα αποτελούν και
// πιθανά ορίσματα των κλήσεων σε συναρτήσεις που πραγματοποιούμε για τη δημιουργία
// των δομών του NacVM.

%type <StringVal> T_ID T_ID_VAR

%token T_LPAREN T_RPAREN T_LBRACE T_RBRACE T_LBRACKET T_RBRACKET
%token T_COMMA T_COLON T_SEMI T_ASSIGN T_EQUAL
%token T_PROCEDURE T_LOCALVAR T_GLOBALVAR T_IN T_OUT
%token T_ID T_ID_VAR

%start nac_top

%%

nac_top : procedure_list
    | globalvar_def procedure_list
    ;

globalvar_def : globalvar_prefix global_id_list T_SEMI
    | globalvar_def globalvar_prefix global_id_list T_SEMI
    ;

globalvar_prefix : T_GLOBALVAR type_spec_global
    ;
```

```
procedure_def : procedure_prefix T_LPAREN arg_list T_RPAREN T_LBRACE stmt_list
                  T_RBRACE {create_lv_range_list(strdup(CUR_PROC_NAME),strdup("no"));}
    | procedure_prefix T_LPAREN arg_list T_RPAREN T_LBRACE localvar_list stmt_list
                T_RBRACE {create_lv_range_list(strdup(CUR_PROC_NAME),strdup("yes"));}
    ;

procedure_list : procedure_def
    | procedure_list procedure_def
    ;

procedure_prefix : T_PROCEDURE id_proc_name
    ;

localvar_list : localvar_prefix local_id_list T_SEMI
    | localvar_list localvar_prefix local_id_list T_SEMI
    ;

localvar_prefix : T_LOCALVAR type_spec_local
    ;

stmt_list : /* empty */
    | stmt_list
    stmt
    ;

stmt : nac
    | pcall
    | label
    ;

nac :  opnd_out_list_nac assign_op id_op opnd_in_list_nac T_SEMI    {gen_stack();}
    | opnd_out_list_nac assign_op id_op T_SEMI                      {gen_stack();}
    | id_op opnd_in_list_nac T_SEMI                                 {gen_stack();}
    | id_op T_SEMI                                                  {gen_stack();}
    ;

pcall   :   T_LPAREN   opnd_out_list_pcall   T_RPAREN   assign_op   id_pcall   T_LPAREN
opnd_in_list_pcall T_RPAREN T_SEMI           {gen_stack();}
| T_LPAREN opnd_out_list_pcall T_RPAREN assign_op id_pcall T_SEMI
                                             {gen_stack();}
    |       id_pcall       T_LPAREN       opnd_in_list_pcall       T_RPAREN       T_SEMI
{gen_stack();}
    |    T_LPAREN    T_RPAREN    assign_op    id_pcall    T_LPAREN    T_RPAREN    T_SEMI
{gen_stack();}
    ;

assign_op : T_ASSIGN
    ;

label :  id_label T_COLON
    ;

opnd_out_list_pcall : id_list_out_pcall
    ;

opnd_out_list_nac : id_list_out_nac
    ;

opnd_in_list_pcall : id_list_in_pcall
    ;

opnd_in_list_nac : id_list_in_nac
    ;

arg_list : /* empty */
    | arg_in
```

```
    | arg_out
    | arg_list T_COMMA arg_in
    | arg_list T_COMMA arg_out
    ;

arg_in : T_IN type_spec id_in
    ;

arg_out : T_OUT type_spec id_out
    ;

global_id_list : global_id
    | global_id_list T_COMMA global_id
    ;

local_id_list : local_id
    | local_id_list T_COMMA local_id
    ;

id_list_out_nac : id_out_nac
    | id_list_out_nac T_COMMA id_out_nac
    ;

id_list_in_nac : id_in_nac
    | id_list_in_nac T_COMMA id_in_nac
    ;

id_list_out_pcall : id_out_pcall
    | id_list_out_pcall T_COMMA id_out_pcall
    ;

id_list_in_pcall : id_in_pcall
    | id_list_in_pcall T_COMMA id_in_pcall
    ;

id_in : T_ID                    {analyze_token_array(strdup($1),'i');}
    ;

id_out : T_ID                   {analyze_token_array(strdup($1),'o');}
    ;

local_id : T_ID                  {analyze_token_array(strdup($1),'l');}
    | T_ID_VAR                   {analyze_token_array(strdup($1),'l');}
    ;

global_id : T_ID                 {analyze_token_array(strdup($1),'g');}
    | T_ID_VAR                   {analyze_token_array(strdup($1),'g');}
    ;

id_pcall : T_ID  {PCALL = '1';if(OPERATION != NULL) free(OPERATION); OPERATION=(char
*)malloc((strlen($1)+1) * sizeof(char));strcpy(OPERATION,$1);}
    ;

id_proc_name : T_ID {if (CUR_PROC_NAME != NULL) free(CUR_PROC_NAME); CUR_PROC_NAME=
(char *)malloc((strlen($1)+1) * sizeof(char));strcpy(CUR_PROC_NAME,$1);}
    ;

id_label : T_ID {if(CUR_LABEL_NAME != NULL) free(CUR_LABEL_NAME); CUR_LABEL_NAME=
(char *)malloc((strlen($1)+1) * sizeof(char));strcpy(CUR_LABEL_NAME,$1);}

id_op : T_ID {if(OPERATION !=NULL)free(OPERATION);OPERATION=(char *)malloc((strlen($1)
+1) * sizeof(char));strcpy(OPERATION,$1)};


id_in_nac : T_ID                                {gen_inarg_list($1);}
    ;
```

```
id_out_nac : T_ID                                   {gen_outarg_list($1);}
    ;

id_in_pcall : T_ID                                  {gen_inarg_list($1);}
    ;

id_out_pcall : T_ID                                 {gen_outarg_list($1);}
    ;

type_spec_global : T_ID {if(GLOBALVAR_TYPE != NULL) free (GLOBALVAR_TYPE);
 GLOBALVAR_TYPE=(char*)malloc((strlen($1)+1)*sizeof (char));strcpy(GLOBALVAR_TYPE,$1)}
    ;

type_spec_local : T_ID {if(LOCALVAR_TYPE != NULL) free(LOCALVAR_TYPE); LOCALVAR_TYPE =
(char *)malloc((strlen($1)+1) * sizeof(char));strcpy(LOCALVAR_TYPE,$1)}
    ;

type_spec  :  T_ID  {if(PROCEDURE_INOUT_TYPE  !=  NULL)  free(PROCEDURE_INOUT_TYPE);
PROCEDURE_INOUT_TYPE  =  (char  *)malloc((strlen($1)+1)  *  sizeof(char);  strcpy(
PROCEDURE_INOUT_TYPE , $1)}
    ;

%%
#include <stdio.h>
#include <stdlib.h>
#include <string.h>    /* For strcmp, strlen  */

extern int column;
extern unsigned long  _LINES_;
extern FILE *yyin;

int yyerror(char *s)
{
  fflush(stdout);
  printf ("%s at line:%ld and column:%d\n",s,_LINES_,column);
  _ERRORS_++;

  return 0;
}
```

3.3 Στοίβα Μηχανής και Πίνακας Συμβόλων

Πριν προχωρήσουμε στην επεξήγηση των συναρτήσεων που εμφανίζονται στους γραμματικούς κανόνες, θα παρουσιάσουμε τη στοίβα μηχανής και τον πίνακα συμβόλων. Η στοίβα μηχανής αποτελεί μία διπλά συνδεδεμένη λίστα με instructions, όπου κάθε κόμβος-instruction περιέχει τις παρακάτω πληροφορίες:

```
struct instructions {
        //Εντολή n-διευθύνσεων (nac) ή κλήση διαδικασίας (pcall)
        char *op;
        //Λίστα με τα input arguments της εντολής ή της κλήσης
        inarg *in;
        //Λίστα με τα output arguments της εντολής ή της κλήσης
        outarg *out;
        //Το όνομα της διαδικασίας στην οποία "ανήκει" η εντολή
        char *procname;
        //Η ετικέτα "μέσα" στην οποία βρίσκεται η εντολή
        char *label;
```

```
    //0 -> nac, 1 -> pcall
    char pcall;
    //Ο αύξων αριθμός του instruction
    unsigned int rank;

    struct instructions *next;
    struct instructions *previous;
};
typedef struct instructions instruction;
instruction *head ,*tail;
```

Όπως γίνεται αντιληπτό, πριν μπορέσουμε να εισάγουμε στη στοίβα ένα νέο κόμβο θα πρέπει να έχουν ήδη δημιουργηθεί οι λίστες με τα ορίσματα εισόδου και εξόδου του τρέχοντος statement. Αποτελούν απλά συνδεδεμένες λίστες και οι κόμβοι τους περιέχουν τις εξής πληροφορίες:

```
struct in_arguments {
    //Όνομα ορίσματος – μεταβλητής εισόδου
    char *arg;
    //Ο αύξων αριθμός του στη λίστα
    unsigned int inrank;
    struct in_arguments *next;
};
typedef struct in_arguments inarg;
inarg *inhead,*intail;

struct out_arguments {
    //Όνομα ορίσματος – μεταβλητής εξόδου
    char *arg;
    //Ο αύξων αριθμός του στη λίστα
    unsigned int outrank;
    struct out_arguments *next;
};
typedef struct out_arguments outarg;
outarg *outhead,*outail;
```

Εδώ λοιπόν αναλαμβάνουν δράση οι συναρτήσεις *gen_inarg_list()* και *gen_outarg_list()* οι οποίες δημιουργούν τις αντίστοιχες λίστες. Ανάλογα λοιπόν ποιος κανόνας επιτυγχάνεται καλείται και η σωστή συνάρτηση. Το αποτέλεσμα είναι πως όταν επαληθευτεί μία nac ή ένα pcall, να έχουν δημιουργηθεί οι λίστες με τα ορίσματα εισόδου και εξόδου τους.

Σε αυτό το σημείο λοιπόν είτε επαληθεύεται ο κανόνας nac: είτε ο κανόνας pcall: (βλέπε επάνω), καλείται η συνάρτηση *gen_stack()*, η οποία προσθέτει ένα νέο κόμβο στη στοίβα. Ο συγχρονισμός για τις τιμές που έχουν οι παρακάτω μεταβλητές του προγράμματος επιτυγχάνεται βάση των κανόνων που είναι εφικτό κάθε φορά να επαληθευτούν βάση της γραμματικής της NAC.

```
char *CUR_PROC_NAME = NULL;
```

```
char *CUR_LABEL_NAME = NULL;
char *OPERATION = NULL;
char PCALL = '0';
```

Έτσι όταν έχει φτάσει η χρονική στιγμή να κληθεί η *gen_stack()*, οι παραπάνω μεταβλητές έχουν τις τρέχουσες και έγκυρες τιμές που επαληθεύει ο συντακτικός αναλυτής. Κάθε φορά στο τέλος της *gen_stack()* καλείται η *initialize_stack()* η οποία αρχικοποιεί τις λίστες των ορισμάτων εισόδου και εξόδου (inarg, outarg) ώστε να είναι έτοιμες να «φιλοξενήσουν» τα ορίσματα της νέας instruction. Πριν βέβαια η συνάρτηση προσθέσει ένα νέο κόμβο στη στοίβα, πραγματοποιεί κάποιους ελέγχους. Με την κλήση της *check_nac_statement()* ελέγχει αν η εντολή που διάβασε ο συντακτικός αναλυτής υπάρχει στη NAC. Αν δεν υπάρχει παράγεται συντακτικό λάθος. Επίσης για να αποφύγουμε αργότερα τις πολλαπλές σαρώσεις τις στοίβας, μέσα από την *gen_stack()*, καλείται η *gen_list_acc_stack_labels()* η οποία δημιουργεί μία πολύ σημαντική βοηθητική λίστα, την proc_lab. Ο κάθε κόμβος της περιέχει τις ακόλουθες πληροφορίες:

```
// Η δομή proc_lab αποτελεί μια δομή που λειτουργεί καταλυτικά στη
//γρήγορη και αποτελεσματική εκτέλεση των jmpzz instructions!
struct procedure_labels{
    //Το όνομα της διαδικασίας
    char *proc_name;
    //Ένας δείκτης σε λίστα label_address
    label_address *la;

    struct procedure_labels *next;
};
typedef struct procedure_labels proc_lab;
proc_lab *pl_head, *pl_tail;
```

Όπως βλέπουμε, το δεύτερο πεδίο του κόμβου proc_lab είναι ένας δείκτης σε λίστα τύπου label_address, ο κάθε κόμβος της οποίας περιέχει το όνομα της ετικέτας και τη διεύθυνση του instruction στη στοίβα όπου εμφανίζεται για πρώτη φορά στη διαδικασία proc_name.

```
struct stack_address_for_labels{
    //Όνομα ετκέτας
    char *label_name;

    //Διεύθυνση κόμβου στη στοίβα όπου εμφανίζεται για πρώτη φορά η
    //ετικέτα label_name στη διαδιακασία proc_name και όχι γενικά στη
    //στοίβα. Καθώς δύο διαφορετικές διαδικασίες μπορούν να έχουν
    //ετικέτες με ίδια ονόματα.
    instruction *address;

    struct stack_address_for_labels *next;
};
```

```
typedef struct stack_address_for_labels label_address;
label_address *la_head, *la_tail;
```

Με αυτό τον τρόπο πετυχαίνουμε χωρίς δεύτερη σάρωση της στοίβας να έχουμε μία δομή που να περιέχει τις ετικέτες της κάθε διαδικασίας και τις διευθύνσεις αυτών στη στοίβα. Αυτή η δομή θα μας φανεί πολύ χρήσιμη στις εντολές της οικογένειας jmpzz (Βλέπε Παράρτημα Α), όπου θα μπορούμε να αλλάζουμε τη ροή του προγράμματος εύκολα και γρήγορα.

Μια ακόμα συνάρτηση που βλέπουμε ότι καλείται στους γραμματικούς κανόνες είναι όταν επαληθεύεται ο κανόνας procedure_def. Η *create_lv_range_list()* δημιουργεί μία δομή απλά συνδεδεμένης λίστας τύπου range_loc όπου ο κάθε κόμβος της περιέχει τις ακόλουθες πληροφορίες:

```
struct range_localvar{
     //Όνομα διαδικασίας
     char *procname;
     //Λίστα με τα ορίσματα εισόδου της διαδικασίας
     proc_inarg *pinarg;
     //Λίστα με τα ορίσματα εξόδου της διαδικασίας
     proc_outarg *poutarg;
     //Λίστα με τις τοπικές μεταβλητές της procname
     localvar_list *lv_list;
     //Ο αύξων αριθμός της τοπικής μεταβλητής στη λίστα
     unsigned long lv_list_rank;

     struct range_localvar *next;
};
typedef struct range_localvar range_loc;
range_loc *range_head, *range_tail;
```

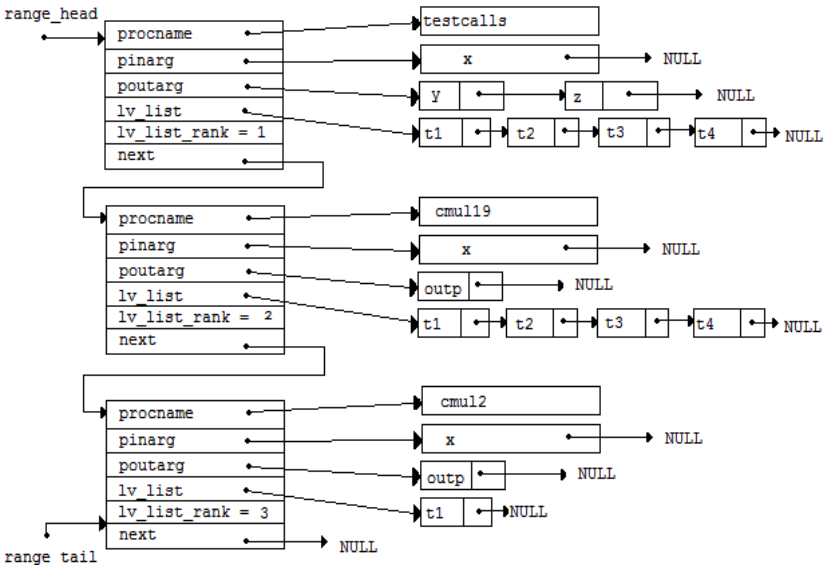
Είναι βασική δομή του προσομοιωτή μας καθώς μας επιτρέπει κάθε φορά να ελέγχουμε, αντιγράφουμε και τροποποιούμε τις τιμές κάθε είδους μεταβλητών. Επίσης μέσα από αυτή τη δομή ελέγχουμε αν οι τοπικές μεταβλητές που χρησιμοποιούνται είναι δηλωμένες και αν βρίσκονται στην εμβέλεια της τρέχουσας διαδικασίας σε ένα πρόγραμμα NAC. Σαν τοπικές μεταβλητές θεωρούνται εκείνες που δηλώνονται σαν localvar καθώς και οι in και out της διαδικασίας. Κάθε διαδικασία του προγράμματος NAC καταλαμβάνει έναν κόμβο αυτής της λίστας. Δηλαδή εάν το αρχείο .nac που προσομοιώνουμε έχει τρεις διαδικασίες, τότε η λίστα αυτή θα αποτελείται από τρεις κόμβους. Παρακάτω ακολουθεί ένα παράδειγμα που δείχνει τα περιεχόμενα της λίστας μετά το τέλος της συντακτικής του ανάλυσης. Έστω για παράδειγμα το ακόλουθο NAC πρόγραμμα

```
// Filename: testcalls.nac
```

```
// Purpose : A simple/synthetic example featuring procedure calls.
// Author  : Nikolaos Kavvadias (C) 2010
// Date    : 30-December-2010
//           0.3.1 (30/12/10)

procedure testcalls (in s32 x, out s32 y, out s32 z)
{
  localvar s32 t1, t2, t3, t4;
S_1:
  t1 <= mov x;
  (t2) <= cmul19 (t1);
  (t3) <= cmul2 (t1);
  t4 <= add t2, t3;
  y <= mov t4;
  z <= mov y;
}
procedure cmul19 (in s32 x, out s32 outp)
{
  localvar s32 t1, t2, t3, t4;
S_1:
  t1 <= mov x;
  t2 <= shl t1, 4;
  t3 <= shl t1, 1;
  t4 <= add t2, t3;
  outp <= add t4, t1;
}
procedure cmul2 (in s32 x, out s32 outp)
{
  localvar s32 t1;
S_1:
  t1 <= mov x;
  outp <= shl t1, 1;
}
```

Τότε μετά τη συντακτική του ανάλυση η δομή range_loc θα έχει τα ακόλουθα δεδομένα



Εικόνα 2. Μέρος της δομής του πίνακα συμβόλων για το testcalls.nac

Και εδώ, όπως προηγουμένως, πριν κληθεί η *create_lv_range_list()* να δημιουργήσει ένα κόμβο range_loc θα πρέπει να έχουν ήδη δημιουργηθεί οι λίστες τύπου proc_inarg, proc_outarg και localvar_list. Αυτό γίνεται μέσω της συνάρτησης *analyze_token_array()*, η οποία δημιουργεί για κάθε διαδικασία τις τρεις παραπάνω λίστες. Οι λίστες δημιουργούνται μέσω της κλήσης κατάλληλων συναρτήσεων μέσα από την *analyze_token_array()*, η οποία θα λέγαμε φιλτράρει έξυπνα τα tokens που δέχεται σαν ορίσματα και τα διοχετεύει στις ανάλογες συναρτήσεις που βρίσκονται στο σώμα της. Εκτός από αυτό, η συγκεκριμένη διαδικασία κάνει και κάτι ακόμα πολύ σημαντικό, αρχικοποιεί τις μεταβλητές-πίνακες, είτε με τις τιμές που δίνουμε εμείς ρητά στο πρόγραμμα μας, είτε τις προεπιλεγμένες τιμές που δίνει ο NacVM. Στο προηγούμενο κεφάλαιο είχαμε αναφέρει πως αν δεν αρχικοποιήσουμε ρητά τις μεταβλητές που δηλώνουμε, ο NacVM τις αρχικοποιεί με την τιμή μηδέν (0).

Πριν προχωρήσουμε όμως και άλλο θα πρέπει να παρουσιάσουμε τις συναρτήσεις που καλούνται μέσα από την *analyze_token_array()* και δημιουργούν τις λίστες proc_inarg, proc_outarg και localvar_list. Η συνάρτηση *put_local_sym()* είναι εκείνη που «χτίζει» τη λίστα localvar_list η οποία για τον κάθε κόμβο της αποθηκεύει τα παρακάτω δεδομένα:

```
struct localvar{
    //Το όνομα της μεταβλητής-πίνακα
    char *varname;
    //Ο τύπος των δεδομένων της μεταβλητής-πίνακα
    char *vartype;
    //Το μέγεθος του πίνακα
    unsigned long array_size;
    //Ένα δείκτη σε δομή τύπου init_array. Θα δούμε στη συνέχεια τι
    //δεδομένα αποθηκεύει και ποια συνάρτηση αναλαμβάνει τη δημιουργία
    //της.
    init_array *array;
    //Ο αύξων αριθμός της μεταβλητής στη λίστα.
    unsigned long lv_rank;

    struct localvar *next;
};
typedef struct localvar localvar_list;
localvar_list *lhead, *ltail;
```

Κι εδώ, πριν προσθέσουμε ένα κόμβο στη λίστα, έχει ήδη δημιουργηθεί μέσω της συνάρτησης *put_lv_array_values()* μια απλά συνδεδεμένη λίστα τύπου init_array η οποία για κάθε μεταβλητή-πίνακα αποθηκεύει τις παρακάτω πληροφορίες:

```
struct array{
    //Τιμή του πίνακα στη θέση index
```

```
    int value;
    //Η θέση του πίνακα στην οποία αντιστοιχεί η παραπάνω τιμή
    unsigned long index;

    struct array *next;
};
typedef struct array init_array;
init_array *arhead, *artail,*arlochead, *arloctail;
```

Στην ουσία κάθε τέτοια λίστα αναπαριστά τις μεταβλητές-πίνακες που δηλώνονται στο πρόγραμμα NAC. Παρατηρήστε στο κώδικα, ότι στο τέλος της *put_local_sym()* καλείται η *initialize_localvar()* η οποία αναλαμβάνει το «μηδενισμό» των δεικτών για να είναι έτοιμοι να δεχθούν τη νέα μεταβλητή-πίνακα και την τιμή ή τις τιμές αυτής.

Επανερχόμαστε σε αυτό το σημείο και συνεχίζουμε μετά την *put_local_sym()*, στη *put_proc_in_args()* η οποία καλείται και αυτή μέσα από την *analyze_token_array()* και αναλαμβάνει τη δημιουργία απλά συνδεδεμένης λίστας τύπου proc_inarg, της οποίας κάθε κόμβος της περιγράφεται όπως παρακάτω:

```
struct proc_in_arguments {
    char *varname;
    char *vartype;
    unsigned long array_size;
    init_array *array;
    unsigned long in_rank;

    struct proc_in_arguments *next;
};
typedef struct proc_in_arguments proc_inarg;
proc_inarg *proc_inhead,*proc_intail;
```

Η λίστα αυτή περιέχει τα ορίσματα εισόδου της κάθε διαδικασίας. Θα παρατηρούσε κάποιος και πολύ σωστά ότι ο τύπος proc_inarg είναι πανομοιότυπος με τον localvar_list. Ο λόγος που επιλέξαμε να δημιουργήσουμε νέο τύπο ήταν αυτός της εύκολης ανάγνωσης και συντήρησης του κώδικα. Για αυτό το λόγο δεν σχολιάσαμε ξανά τα πεδία παραπάνω. Επειδή και αυτές οι μεταβλητές θεωρούνται τοπικές, για τη δημιουργία της λίστας τύπου init_array χρησιμοποιείται πάλι η *put_lv_array_values()* . Τέλος, τη δημιουργία της λίστας με τα ορίσματα εξόδου της διαδικασίας την αναλαμβάνει η *put_proc_out_args()*. Και εδώ ο τύπος του κόμβου είναι παρόμοιος με τους παραπάνω:

```
struct proc_out_arguments {
    char *varname;
    char *vartype;
    unsigned long array_size;
    init_array *array;          //Κι εδώ τη λίστα τη δημιουργεί η
                                // put_lv_array_values()
```

```
        unsigned long out_rank;

        struct proc_out_arguments *next;
    };
    typedef struct proc_out_arguments proc_outarg;
    proc_outarg *proc_outhead,*proc_outail;
```

Επιστρέφουμε τώρα στη *create_lv_range_list()* η οποία αφού έχουν δημιουργηθεί οι απαραίτητες λίστες, προσθέτει ένα νέο κόμβο στη λίστα range_loc. Πριν κάποια μεταβλητή-πίνακας εισαχθεί στη range_loc ελέγχεται ο τύπος και το μέγεθός της μέσω της συνάρτησης *check_type_size()*. Από όλα τα παραπάνω προκύπτει ότι η NAC υποστηρίζει εντολές n-διευθύνσεων, ότι κάθε μεταβλητή μπορεί να είναι μονοδιάστατος πίνακας με μέγεθος μέχρι INT_MAX και ότι το πλήθος των ορισμάτων εισόδου και εξόδου της κάθε διαδικασίας μπορεί να φτάσει το μέγεθος της μνήμης του υπολογιστή.

Σημαντικό: Η μία και μοναδική δομή τύπου range_loc που δημιουργείται για κάθε πρόγραμμα NAC που εκτελεί ο NacVM αποτελεί τον πίνακα συμβόλων του προσομοιωτή. Αλλά δεν είναι από μόνη της ο πίνακας συμβόλων, καθώς δεν έχουμε αναφέρει καθόλου τις καθολικές μεταβλητές.

Η δομή με τις καθολικές μεταβλητές λοιπόν δημιουργείται και αυτή μέσω της *analyze_token_array()* και της *put_global_sym()*. Είναι τύπου globalvar_list και κάθε κόμβος της περιέχει τις ακόλουθες πληροφορίες,

```
struct globalvar{
     char *varname;
     char *vartype;
     unsigned long array_size;
     init_array *array;
     unsigned long gv_rank;

     struct globalvar *next;
};
typedef struct globalvar globalvar_list;
globalvar_list *ghead, *gtail;
```

οι οποίες είναι παρόμοιες με αυτές των τριών προηγούμενων τύπων δεδομένων. Εδώ η δομή τύπου init_array δομείται από τη συνάρτηση *put_gv_array_values()*. Επομένως οι δύο αυτές δομές (range_loc, globalvar_list) αποτελούν τον πίνακα συμβόλων του NacVM. Ακολουθεί ένα παράδειγμα με τα στοιχεία και τον τρόπο που αποθηκεύει τα δεδομένα του ο πίνακας συμβόλων όταν εκτελείται ένα πρόγραμμα NAC. Το παράδειγμα αφορά την υλοποίηση του αλγορίθμου ταξινόμησης bubblesort.

Στο συγκεκριμένο παράδειγμα αρχικοποιούμε ένα πίνακα είκοσι (20) θέσεων και στη συνέχεια τον ταξινομούμε.

```
globalvar  s32  table[20]={3123,-9,331,894,442,67,13,-69,  144,8678976,321,88,
0,771,1278,53,-1000,541,900,-5}, N[1]={20};
procedure bubblesort (in s32 a, out s32 b)
{
       localvar s32 t1, t2, t3, t4, pos, pos2, i;
       B:
              pos <= mov -1;
              i <= add i, 1;
              L, S_EXIT <= jmplt i, N;
       L:
              pos <= add pos, 1;
              t4 <= sub N, i;
              L_1, B <= jmplt pos, t4;
       L_1:
              t1 <= load table, pos;
              pos2 <= mov pos;
              pos <= add pos, 1;

              t2 <= load table, pos;
              pos <= sub pos, 1;
              L_2, L <= jmpgt t1, t2;
       L_2:
           table <= store t2, pos2;
              pos2 <= add pos2, 1;
              table <= store t1, pos2;
              L <= jmpun;
       S_EXIT:
              print table;
}
```

Η εκτέλεση του παραπάνω προγράμματος θα έχει σαν έξοδο τα παρακάτω:

```
table:
Hex:[fffffc18] - Dec:[-1000]
Hex:[ffffffbb] - Dec:[-69]
Hex:[fffffff7] - Dec:[-9]
Hex:[fffffffb] - Dec:[-5]
Hex:[0] - Dec:[0]
Hex:[d] - Dec:[13]
Hex:[35] - Dec:[53]
Hex:[43] - Dec:[67]
Hex:[58] - Dec:[88]
Hex:[90] - Dec:[144]
Hex:[141] - Dec:[321]
Hex:[14b] - Dec:[331]
Hex:[1ba] - Dec:[442]
Hex:[21d] - Dec:[541]
Hex:[303] - Dec:[771]
Hex:[37e] - Dec:[894]
Hex:[384] - Dec:[900]
Hex:[4fe] - Dec:[1278]
Hex:[c33] - Dec:[3123]
Hex:[846e40] - Dec:[8678976]

**********NacVM RESULTS*********
_____________________________
Nac File: [tests/bubblesort.nac]
Input Data File:
[tests/bubblesort_test_data.txt]

1: Correct

-Elapsed time: 0.015 seconds.
```

Εικόνα 3. Αποτελέσματα εκτέλεσης σεναρίου του bubblesort.nac

Για να δούμε όμως τα περιεχόμενα του πίνακα συμβόλων θα εκτελέσουμε δύο βοηθητικές συναρτήσεις, την *print_globalvar()* και την *print_localvar_for_list().*

```
______GLOBALVAR______

Node:      [1]
VarName:   [table]
VarType:   [s32]
ArraySize: [20]
Index - Value
[0] - [3123]
[1] - [-9]
[2] - [331]
[3] - [894]
[4] - [442]
[5] - [67]
[6] - [13]
[7] - [-69]
[8] - [144]
[9] - [8678976]
[10] - [321]
[11] - [88]
[12] - [0]
[13] - [771]
[14] - [1278]
[15] - [53]
[16] - [-1000]
[17] - [541]
[18] - [900]
[19] - [-5]


Node:      [2]
VarName:   [N]
VarType:   [s32]
ArraySize: [1]
Index - Value
[0] - [20]


______PROCEDURE ARGUMENTS & LOCAL VARIABLES______

Procedure Name: [bubblesort]
Procedure InOut Arguments: ->

Node-In-Args: [1]
VarName:      [a]
VarType:      [s32]
ArraySize:    [1]
Index - Value
[0] - [0]

Node-Out-Args: [1]
VarName:       [b]
VarType:       [s32]
ArraySize:     [1]
Index - Value
[0] - [0]

________________
Local Variables: ->
________________

Node:      [1]
VarName:   [t1]
```

```
VarType:   [s32]
ArraySize: [1]
Index - Value
[0] - [0]

Node:      [2]
VarName:   [t2]
VarType:   [s32]
ArraySize: [1]
Index - Value
[0] - [0]

Node:      [3]
VarName:   [t3]
VarType:   [s32]
ArraySize: [1]
Index - Value
[0] - [0]

Node:      [4]
VarName:   [t4]
VarType:   [s32]
ArraySize: [1]
Index - Value
[0] - [0]

Node:      [5]
VarName:   [pos]
VarType:   [s32]
ArraySize: [1]
Index - Value
[0] - [0]

Node:      [6]
VarName:   [pos2]
VarType:   [s32]
ArraySize: [1]
Index - Value
[0] - [0]

Node:      [7]
VarName:   [i]
VarType:   [s32]
ArraySize: [1]
Index - Value
[0] - [0]
```

Επειδή παραπάνω παρουσιάσαμε τη στοίβα μηχανής καλό θα ήταν να δείξουμε και τα περιεχόμενα της για το τελευταίο παράδειγμα. Έτσι θα καλέσουμε την *print_stack()*.

```
______STACK______

Node:      [1]
Procedure: [bubblesort]
PCALL:     [0]
Label:     [B]
Operation: [mov]
Out Args:  [pos]
In Args:   [-1]

Node:      [2]
Procedure: [bubblesort]
PCALL:     [0]
Label:     [B]
Operation: [add]
```

```
Out Args:  [i]
In Args:   [i] - [1]

Node:      [3]
Procedure: [bubblesort]
PCALL:     [0]
Label:     [B]
Operation: [jmplt]
Out Args:  [L] - [S_EXIT]
In Args:   [i] - [N]

Node:      [4]
Procedure: [bubblesort]
PCALL:     [0]
Label:     [L]
Operation: [add]
Out Args:  [pos]
In Args:   [pos] - [1]

Node:      [5]
Procedure: [bubblesort]
PCALL:     [0]
Label:     [L]
Operation: [sub]
Out Args:  [t4]
In Args:   [N] - [i]

Node:      [6]
Procedure: [bubblesort]
PCALL:     [0]
Label:     [L]
Operation: [jmplt]
Out Args:  [L_1] - [B]
In Args:   [pos] - [t4]

Node:      [7]
Procedure: [bubblesort]
PCALL:     [0]
Label:     [L_1]
Operation: [load]
Out Args:  [t1]
In Args:   [table] - [pos]

Node:      [8]
Procedure: [bubblesort]
PCALL:     [0]
Label:     [L_1]
Operation: [mov]
Out Args:  [pos2]
In Args:   [pos]

Node:      [9]
Procedure: [bubblesort]
PCALL:     [0]
Label:     [L_1]
Operation: [add]
Out Args:  [pos]
In Args:   [pos] - [1]

Node:      [10]
Procedure: [bubblesort]
PCALL:     [0]
Label:     [L_1]
Operation: [load]
Out Args:  [t2]
In Args:   [table] - [pos]
```

```
Node:      [11]
Procedure: [bubblesort]
PCALL:     [0]
Label:     [L_1]
Operation: [sub]
Out Args:  [pos]
In Args:   [pos] - [1]

Node:      [12]
Procedure: [bubblesort]
PCALL:     [0]
Label:     [L_1]
Operation: [jmpgt]
Out Args:  [L_2] - [L]
In Args:   [t1] - [t2]

Node:      [13]
Procedure: [bubblesort]
PCALL:     [0]
Label:     [L_2]
Operation: [store]
Out Args:  [table]
In Args:   [t2] - [pos2]

Node:      [14]
Procedure: [bubblesort]
PCALL:     [0]
Label:     [L_2]
Operation: [add]
Out Args:  [pos2]
In Args:   [pos2] - [1]

Node:      [15]
Procedure: [bubblesort]
PCALL:     [0]
Label:     [L_2]
Operation: [store]
Out Args:  [table]
In Args:   [t1] - [pos2]

Node:      [16]
Procedure: [bubblesort]
PCALL:     [0]
Label:     [L_2]
Operation: [jmpun]
Out Args:  [L]
In Args:   [(null)]

Node:      [17]
Procedure: [bubblesort]
PCALL:     [0]
Label:     [S_EXIT]
Operation: [print]
Out Args:  [(null)]
In Args:   [table]
```

Όλες οι παραπάνω διαδικασίες προκαλούνται από την κλήση της μεθόδου *nacparse()* η οποία βρίσκεται στη *main()* στο αρχείο nacparse.c.

3.4 Βοηθητικές Δομές και Χρήσιμες Συναρτήσεις

Μετά τη *nacparse()* ακολουθεί η *create_proc_list()* η οποία δημιουργεί μία βοηθητική δομή χρήσιμη για τις κλήσεις διαδικασιών στα προγράμματα NAC και για το σωστό διάβασμα του αρχείου των δεδομένων εισόδου. Η δομή είναι μία απλά συνδεδεμένη λίστα τύπου proc_list_acc η οποία σε κάθε κόμβο περιέχει πληροφορίες (όνομα διαδικασίας, διεύθυνσή της στη στοίβα, πλήθος ορισμάτων (in-out)) για κάθε διαδικασία του προγράμματος. Έτσι όταν στη *fetch_execute_cycle()* έχουμε κλήση διαδικασίας, αυτή η δομή θα μας βοηθάει να "μεταβαίνουμε" και να "επιστρέφουμε" γρήγορα στη σωστή διεύθυνση της στοίβας. Το μειονέκτημα σε αυτή τη δομή είναι ότι απαιτείται μία δεύτερη σάρωση στη στοίβα για να δημιουργηθεί.

```
struct proc_list{
    char *procname;
    instruction *address;
    unsigned short mass_of_in_args;
    unsigned short mass_of_out_args;

    struct proc_list *next;
};
typedef struct proc_list proc_list_acc;
proc_list_acc *proc_head, *proc_tail;
```

Ακολουθεί στη *main()* η κλήση της *read_data_from_file()* η οποία δημιουργεί μία δομή που «φιλοξενεί» τα δεδομένα εισόδου του προγράμματος NAC που σκοπεύουμε να προσομοιώσουμε. Ο τύπος της δομής είναι file με τις παρακάτω πληροφορίες,

```
struct input_data_from_file_to_nacvm {
    //Ένας δείκτης στη λίστα με τα δεδομένα εισόδου της γονικής
    //διαδικασίας
    ina *in;
    //Ένας δείκτης στη λίστα με τα δεδομένα εξόδου της γονικής
    //διαδικασίας. Οι σωστές τιμές που θα έπρεπε να υπολογίσει ο NacVM.
    outa *out;
    //Το πλήθος των γραμμών του αρχείου με τα δεδομένα
    unsigned long number_of_lines;

    struct input_data_from_file_to_nacvm *next;
};
typedef struct input_data_from_file_to_nacvm file;
file *file_head = NULL, *file_tail = NULL;
```

Οι λίστες τύπου ina και outa δημιουργούνται μέσα στην ίδια συνάρτηση και οι κόμβοι τους έχουν τις ακόλουθες πληροφορίες:

```
struct input_args{
     int in_arg_value;
     unsigned long in_arg_rank;
     struct input_args *next;
};
typedef struct input_args ina;
ina *in_args_head, *in_args_tail;
```

```
struct output_args{
    int out_arg_value;
    unsigned long out_arg_rank;
    struct output_args *next;
};
typedef struct output_args  outa;
outa *out_args_head, *out_args_tail;
```

Τα δεδομένα διαβάζονται σε δεκαεξαδική μορφή για αυτό και στο αρχείο θα πρέπει να έχουν τη μορφή αυτή. Η *read_data_from_file()* γνωρίζει μέσω της δομής τύπου proc_list_acc, που αναφέραμε παραπάνω, το πλήθος των ορισμάτων εισόδου και εξόδου της γονικής διαδικασίας και με αυτό τον τρόπο «γεμίζει» με τα σωστά δεδομένα τη λίστα file.

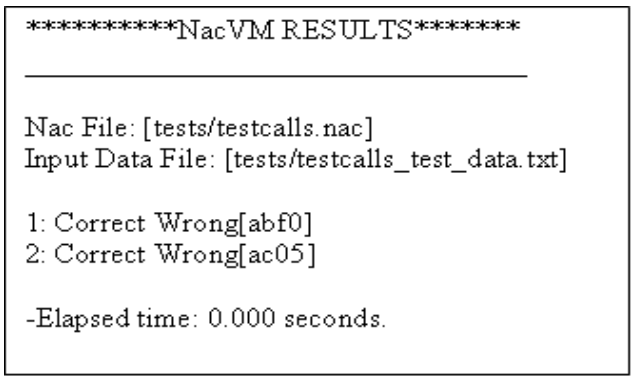
Προσοχή! Εμείς από τη μεριά μας όταν φτιάχνουμε το αρχείο με τα δεδομένα δοκιμής θα πρέπει να έχουμε υπόψη μας το πλήθος των ορισμάτων (εισόδου-εξόδου) της γονικής διαδικασίας του προγράμματος που πρόκειται να προσομοιώσουμε. Η μορφή που θα πρέπει να έχει το αρχείο είναι η ακόλουθη: Πρώτα γράφουμε με διατεταγμένη σειρά τις τιμές εισόδου και στη συνέχεια στην ίδια γραμμή τις τιμές εξόδου. Πρέπει να υπάρχει ένα-προς-ένα αντιστοίχηση των ορισμάτων με αυτά των δεδομένων. Όλες οι τιμές θα πρέπει να είναι διαχωρισμένες με ένα whitespace (space or tab). Θα πρέπει να ξέρουμε τις τιμές εξόδου με βάση αυτές τις εισόδου, εφόσον θέλουμε να διαπιστώσουμε αν ο NacVM λειτουργεί σωστά. Επίσης η τελευταία γραμμή του αρχείου αυτού πρέπει να είναι κενή. Υπάρχει η περίπτωση όμως το πρόγραμμα μας να μην χρειάζεται κάποιες τιμές εισόδου για να λειτουργήσει, όπως στο παραπάνω παράδειγμα της bubblesort. Τότε στο αρχείο εισόδου δεδομένων βάζουμε στις αντίστοιχες θέσεις το μηδέν. Για κάθε γραμμή του αρχείου δεδομένων ο NacVM εκτελεί ένα σενάριο προσομοίωσης του προγράμματος NAC που έχουμε εισάγει.

Ένα παράδειγμα αρχείου δεδομένων δοκιμής για το πρόγραμμα testcalls που παρουσιάσαμε παραπάνω θα μπορούσε να ήταν το παρακάτω:

00000830 0000abf0 0
00000831 0000ac05 0

Η γονική διαδικασία της testcalls βλέπουμε ότι δέχεται μία είσοδο και παράγει δύο εξόδους. Άρα στο πρώτο σενάριο έχουμε x = 00000830 και ισχυριζόμαστε ότι οι τιμές εξόδου για τη y και τη z θα πρέπει να είναι 0000abf0 και 0 αντίστοιχα. Αφού

εκτελέσαμε το testcalls.nac με δεδομένα δοκιμής τα παραπάνω βλέπουμε αυτό που απάντησε ο NacVM:



Εικόνα 4. Αποτελέσματα εκτέλεσης δύο σεναρίων του testcalls.nac

Το Nac File είναι το πρόγραμμα που προσομοιώνουμε ενώ το Input Data File είναι το αρχείο με τα δεδομένα δοκιμής. Ακολουθούν τα αποτελέσματα όπου το «1: Correct Wrong[abf0]» σημαίνει ότι για το πρώτο σενάριο, δηλαδή x = 00000830, το y = 0000abf0 είναι σωστό (δηλαδή Correct), ενώ το z = 0 είναι λάθος (Wrong) και ότι η σωστή του τιμή είναι η z = 0000abf0. Λογικό, αφού στη τελευταία γραμμή της testcalls εκτελούμε z <= mov y;

Συμπεραίνουμε λοιπόν ότι ο NacVM μπορεί και να υπολογίσει τις τιμές εξόδου ενός προγράμματος. Αν για παράδειγμα δεν γνωρίζουμε τα αποτελέσματα ενός προγράμματος NAC μπορούμε απλά στα σενάρια, να δίνουμε στα ορίσματα εξόδου την τιμή μηδέν (0). Με αυτό τον τρόπο θα μας απαντάει Wrong για κάθε έξοδο, αλλά δίπλα και μέσα σε αγκύλες θα εμφανίζει τη σωστή τιμή σε δεκαεξαδική μορφή. Τέλος εμφανίζει το χρόνο σε δευτερόλεπτα που χρειάστηκε για να προσομοιώσει όλα τα σενάρια. Παρατηρούμε ότι στο συγκεκριμένο παράδειγμα ο χρόνος ήταν κάτω από ένα millisecond.

Η τελευταία συνάρτηση που καλείται στη *main()* είναι η *fetch_execute_cycle()* η οποία αποτελεί και την καρδιά του NacVM. Βρίσκεται στη βιβλιοθήκη stack_mac_nac.h και λειτουργεί χρησιμοποιώντας τη στοίβα μηχανής, τον πίνακα συμβόλων καθώς και κάποιες βοηθητικές δομές. Ξεκινώντας έχουμε:

```
…
while(fh -> next != NULL)
{
   if ((pass_by_value_in(fh->in)) != 0)
   {
      fprintf(stderr,"\n Unable to pass the input values from file:
      [%s].\n", INPUT_DATA_FNAME);
      exit(-1);
```

```
   }
…
```

Το εξωτερικό αυτό while αφορά τα σενάρια που έχουν φορτωθεί στη δομή τύπου file. Οι τιμές εισόδου του σεναρίου θα πρέπει να περάσουν στη γονική διαδικασία του NAC προγράμματος. Αυτό γίνεται με την κλήση της διαδικασίας *pass_by_value_in()* η οποία δέχεται σαν όρισμα τη λίστα με τις τιμές εισόδου του τρέχοντος σεναρίου κάθε φορά. Το επόμενο while που συναντάμε αφορά τη σάρωση της στοίβας της μηχανής.

```
pc = head;
while(1)
{
   if (pc == NULL)
   {
      if (clist_tail -> to_procedure != NULL)
      {
         pc = clist_tail -> from_address;
         copy_proc_out_args(clist_tail->to_procedure,pc->procname,pc-> out);
         pc = pc -> next;
         if (clist_tail == clist_head)
             clist_tail -> to_procedure = NULL;
         else
             clist_tail = clist_tail -> previous;
       }
       //Αν η διαδικασία που κλήθηκε είναι η τελευταία του
       //αρχείου του κώδικα.
       else
          break;
     }
     else
     {
        if((strcmp(pc->procname,head->procname)!=0)&&clist_tail->
        to_procedure == NULL)
        {
           pc = tail;
           break;
        }
        if (clist_tail -> to_procedure != NULL)
        {
           if (strcmp(pc -> procname, clist_tail -> to_procedure) != 0)
           {
              pc = clist_tail -> from_address;
              copy_proc_out_args(clist_tail -> to_procedure, pc -> procname,
              pc -> out);
              pc = pc -> next;

              if (clist_tail == clist_head)
                 clist_tail -> to_procedure = NULL;
              else
                 clist_tail = clist_tail -> previous;
           }
         }
      }
```

Πριν προχωρήσει ο δείκτης στον επόμενο κόμβο της στοίβας γίνονται κάποιοι περίπλοκοι έλεγχοι με σκοπό να ανιχνεύσουμε κυρίως το τέλος της γονικής διαδικασίας στο αρχείο. Επίσης ένα άλλο πρόβλημα που αντιμετωπίζει είναι όταν ο

δείκτης της στοίβας φτάνει στο τέλος της (με κλήση διαδικασίας), χωρίς όμως η γονική διαδικασία να έχει ολοκληρωθεί. Τέλος ελέγχει πότε η διαδικασία που κλήθηκε έφτασε στο τέλος της και τότε πρέπει να επιστρέψει τις τιμές εξόδου της στο σημείο που έγινε η κλήση της. Το πέρασμα των τιμών γίνεται μέσω της *copy_proc_out_args()* η οποία ενεργεί πάνω στις δομές του πίνακα συμβόλων. Θυμίζουμε ότι είναι δυνατό το μέγεθος των μεταβλητών-πινάκων εξόδου να είναι μεγαλύτερο του ένα. Δηλαδή η διαδικασία να επιστρέφει έναν πίνακα δεδομένων.

Αφού γίνουν οι παραπάνω έλεγχοι, ελέγχουμε στη συνέχεια αν τα τρέχοντα περιεχόμενα της στοίβας αφορούν εντολή n-διευθύνσεων ή κλήση διαδικασίας. Στη περίπτωση που έχουμε εντολή, ελέγχεται ποια εντολή είναι μέσω της δομής if-then-else και καλείται η κατάλληλη συνάρτηση του NacVM (Παράρτημα Α) που αντιστοιχεί στην εντολή της γλώσσας NAC. Δύο συναρτήσεις που χρησιμοποιούνται σε αυτό το επίπεδο είναι η *is_number()* και η *value()*. Η πρώτη ελέγχει αν το όρισμα εισόδου της εντολής είναι ένας ακέραιος προσημασμένος αριθμός ή είναι ένα όνομα μεταβλητής-πίνακα. Αυτό συμβαίνει γιατί η στοίβα αποθηκεύει όλα τα δεδομένα σαν συμβολοσειρές. Αν είναι αριθμός το όρισμα, τότε επιστρέφει τον αριθμό αυτό, διαφορετικά επιστρέφεται η τιμή INT_MIN. Κάναμε τη θεώρηση ότι πρόκειται για μια τιμή που έχει πάρα πολύ μικρή πιθανότητα να χρησιμοποιηθεί σε κάποιο σενάριο. Η *value()* τώρα είναι εκείνη η συνάρτηση που εντοπίζει στις δομές του πίνακα συμβόλων την τιμή της εκάστοτε μεταβλητής-πίνακα και επιστρέφει ένα δείκτη σε αυτή. Σε αυτή την έκδοση ο δείκτης δείχνει πάντα στη θέση μηδέν του πίνακα, ανεξάρτητα από το μέγεθος του. Ανάλογα λοιπόν με τη μορφή των ορισμάτων των εντολών της NAC καλείται κάθε φορά και η εκάστοτε συνάρτηση του NacVM με τα ανάλογα ορίσματα. Στο προηγούμενο Κεφάλαιο αναφέραμε αναλυτικά ποιες εντολές επιτρέπουν σαν όρισμα εισόδου κυριολεκτικό προσημασμένο αριθμό και σε ποιες θέσεις.

Στη περίπτωση που το statement αφορά κλήση διαδικασίας στη NAC τότε εκτελούνται με τη σειρά οι παρακάτω συναρτήσεις:

- *checks_if_proc_exists();*
- *find_address();*
- *copy_proc_in_args();*
- *do_call();*

Η πρώτη ελέγχει αν η διαδικασία που επιχειρείται να κληθεί υπάρχει στο πρόγραμμα. Η *find_address()* με βάση το όνομα της διαδικασίας που καλείται, επιστρέφει ένα δείκτη σε κόμβο της δομής τύπου proc_list_acc, ο οποίος ανάμεσα στις άλλες πληροφορίες που έχει είναι και η διεύθυνση της διαδικασίας στη στοίβα. Στη συνέχεια η *copy_proc_in_args()* αναλαμβάνει το πέρασμα των τιμών εισόδου της κλήσης στα ορίσματα εισόδου της διαδικασίας. Κι εδώ, όπως στα ορίσματα εξόδου, το μέγεθος του πίνακα μπορεί να είναι μεγαλύτερο του ένα. Στο τέλος καλείται η *do_call()* η οποία δημιουργεί μία πολύ χρήσιμη δομή τύπου clist:

```
//Η δομή clist περιέχει σε κάθε κόμβο τη διεύθυνση της στοίβας ΑΠΟ την οποία
//γίνεται η κλήση της διαδικασίας, αλλά και τη διεύθυνση της στοίβας στην
//οποία μεταβαίνει η ροή του προγράμματος. Επίσης περιέχει το όνομα της
//procedure από την οποία γίνεται η κλήση και το όνομα της procedure που
//καλείται.

   struct call_list{
        instruction *from_address;
        instruction *to_address;
        char *from_procedure;
        char *to_procedure;

        struct call_list *next;
        struct call_list *previous;
    };
    typedef struct call_list clist;
    clist *clist_head,*clist_tail;
```

Αυτή η δομή είναι πολύ χρήσιμη στη *fetch_execute_cycle()* καθώς λύνει το πρόβλημα των πολλαπλών «εμφωλευμένων» κλήσεων διαδικασιών. Στην επόμενη ενότητα θα παρουσιάσουμε και παράδειγμα με πολλαπλές κλήσεις διαδικασιών. Τέλος όταν ολοκληρωθεί ένα σενάριο, δηλαδή ο δείκτης της στοίβας φτάσει στο τέλος της, τότε εκτελείται η *pass_by_value_out()* η οποία καλεί με τη σειρά της την *check_output_value()* περνώντας της τα κατάλληλα ορίσματα. Η τελευταία τώρα αναλαμβάνει να συγκρίνει την τιμή ή τις τιμές εξόδου που υπολόγισε ο NacVM με αυτές του αρχείου δεδομένων δοκιμής, που έχουν φορτωθεί κατά την αρχή εκτέλεσης στη δομή file. Τα αποτελέσματα αυτής της σύγκρισης φαίνονται στο τέλος όλα μαζί.

4. Παραδείγματα NAC Προγραμμάτων

Σε αυτό εδώ το κεφάλαιο θα σας παρουσιάσουμε NAC προγράμματα και τα αποτελέσματα που παράγει ο NacVM σε διάφορα σενάρια εκτέλεσης τους. Επίσης θα μπορέσουμε να εξετάσουμε τις δυνατότητες του NacVM καθώς και τι θα πρέπει να

προσέχουμε όταν γράφουμε τα προγράμματά μας. Πριν προχωρήσουμε θα θέλαμε να επαναλάβουμε τα τρία δυνατά σημεία της NAC που την κάνει να ξεχωρίζει.

1) Δυνατότητα υποστήριξης εντολών n-διευθύνσεων.
2) Υποστήριξη κλήσεων διαδικασιών με n πλήθος ορισμάτων εισόδου και εξόδου.
3) Υποστήριξη μονοδιάστατων πινάκων μεγέθους μέχρι INT_MAX.

Σε όλα τα παρακάτω παραδείγματα, εκτός από το πρώτο, θα είναι εύκολο να εντοπίσετε και τα τρία ιδιαίτερα χαρακτηριστικά της γλώσσας. Το πρώτο παράδειγμα, αφορά τον υπολογισμό της ακολουθίας Fibonacci, είναι γραμμένο από τον Νικόλαο Καββαδία και πρόκειται να το προσομοιώσουμε δίνοντας του ένα αρχείο με δεδομένα δοκιμής είκοσι σεναρίων. Τα δεδομένα αυτά φαίνονται στο εμφωλευμένο πλαίσιο σε δεκαεξαδική μορφή. Η πρώτη στήλη αφορά δεδομένα εισόδου και η δεύτερη εξόδου.

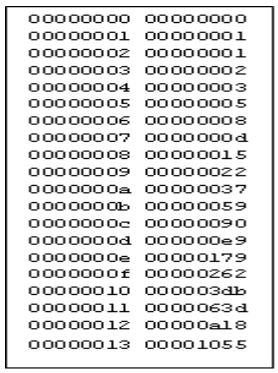
```
// Filename: fibo.nac
// Purpose : N-address code (NAC) implementation for the iterative version of
//           Fibonacci series computation.
// Author  : Nikolaos Kavvadias (C) 2009, 2010
// Date    : 17-Sep-2009
// Revision: 0.2.0 (17/09/09)

procedure fibo(in s32 n, out s32 outp)
{
  localvar s32 res, x;
  localvar s32 f0, f1, f, k;

LL0:
  x <= mov n;
  f0 <= ldc 0;
  f1 <= ldc 1;
  res <= mov f0;
  S_EXIT, LL1 <= jmple x, 0;
LL1:
  res <= mov f1;
  S_EXIT, LL2 <= jmpeq x, 1;
LL2:
  k <= ldc 2;
  LL3 <= jmpun;
LL3:
  f <= add f1, f0;
  f0 <= mov f1;
  f1 <= mov f;
  res <= mov f;
  k <= add k, 1;
  LL3, S_EXIT <= jmple k, x;
S_EXIT:
  outp <= mov res;
}
```

Εικόνα 5. Υπολογισμός ακολουθίας Fibonacci.nac (fibo.nac).

Εκτελούμε τον NacVM δίνοντας του ως πρώτο όρισμα το αρχείο fibo.nac που περιέχει τον κώδικα εκτέλεσης-προσομοίωσης και σαν δεύτερο όρισμα το fibo_test_data.txt (Εικόνα 6.) που περιέχει τα δεδομένα δοκιμής. Τα αποτελέσματα φαίνονται στον αριστερό πίνακα της Εικόνας 7.:



```
00000000 00000000
00000001 00000001
00000002 00000001
00000003 00000002
00000004 00000003
00000005 00000005
00000006 00000008
00000007 0000000d
00000008 00000015
00000009 00000022
0000000a 00000037
0000000b 00000059
0000000c 00000090
0000000d 000000e9
0000000e 00000179
0000000f 00000262
00000010 000003db
00000011 0000063d
00000012 00000a18
00000013 00001055
```

Εικόνα 6. fibo_test_data.txt

```
**********NacVM RESULTS**********
_____________________________
Nac File:[tests/fibo.nac]
Input Data File:
[tests/fibo_test_data.txt]

1: Correct
2: Correct
3: Correct
4: Correct
5: Correct
6: Correct
7: Correct
8: Correct
9: Correct
10: Correct
11: Correct
12: Correct
13: Correct
14: Correct
15: Correct
16: Correct
17: Correct
18: Correct
19: Correct
20: Correct

-Elapsed time: 0.000 seconds.
```

```
**********NacVM RESULTS**********
_____________________________
Nac File: [tests/fibo.nac]
Input Data File:
[tests/fibo_test_data.txt]

1: Correct
2: Wrong[1]
3: Wrong[1]
4: Wrong[2]
5: Wrong[3]
6: Wrong[5]
7: Wrong[8]
8: Wrong[d]
9: Wrong[15]
10: Wrong[22]
11: Wrong[37]
12: Wrong[59]
13: Wrong[90]
14: Wrong[e9]
15: Wrong[179]
16: Wrong[262]
17: Wrong[3db]
18: Wrong[63d]
19: Wrong[a18]
20: Wrong[1055]

-Elapsed time: 0.016 seconds.
```

Εικόνα 7. Αποτελέσματα εκτέλεσης διάφορων σεναρίων του fibo.nac. Αρχείο δεδομένων δοκιμής fibo_test_data.txt.

Αν είχαμε στη διάθεση μας μόνο τα δεδομένα εισόδου και θέλαμε να δούμε τις εξόδους του fibo.nac, τότε στη δεξιά στήλη fibo_test_data.txt θα έπρεπε να βάλουμε σε όλα τα σενάρια την τιμή μηδέν. Τότε τα αποτελέσματα θα ήταν αυτά που φαίνονται στην δεξιά πλευρά της Εικόνας 7. Παρατηρήστε ότι ο χρόνος προσομοίωσης αγγίζει το πολύ τα 16 milliseconds και για τα είκοσι σενάρια.

Τα επόμενα δύο παραδείγματα είναι πιο «απαιτητικά» όσον αφορά τον NacVM και χρησιμοποιούν περισσότερες δυνατότητες της γλώσσας. Και τα δύο έχουν γραφτεί από τον Νικόλαο Καββαδία.

```
globalvar s32 arr[10]={2,3,5,7,11,13,17,19,23,29};

procedure arrayargs (in s32 n, out s32 s)
{
  localvar s32 g, i, t, t0, arr2[10];
S_1:
  g <= mov n;
  t <= ldc 0;
  S_2 <= jmpun;

S_2:
  (arr2) <= func1 (arr);
  S_3 <= jmpun;
S_3:
  i <= ldc 0;
  S_EXIT, S_4 <= jmpeq g, 0;
S_4:
  S_5, S_EXIT <= jmplt i, 10;
S_5:
  t0 <= load arr2, i;
  t <= add t, t0;
  i <= add i, 1;
  S_4, S_EXIT <= jmplt i, g; S_EXIT:
  s <= mov t;
}
procedure func1 (in s32 b[10], out s32 c[10])
{
  localvar s32 i, t;
  localvar s32 t0;
S_1:
  i <= ldc 0;
  S_2 <= jmpun;
S_2:
  t <= load b, i;
  t0 <= add t, 0;
  c <= store t0, i;
  i <= add i, 1;
  S_2, S_EXIT <= jmplt i, 10;
S_EXIT:
  i <= mov i;
}
```

Εικόνα 8. Κλήση διαδικασίας με in-out ορίσματα μονοδιάστατους πίνακες (arrayargs.nac).

Παρατηρήστε ότι στην ετικέτα S_2, γίνεται κλήση της func1 με όρισμα εισόδου τον πίνακα arr μεγέθους δέκα θέσεων και όρισμα εξόδου τον πίνακα arr2 επίσης δέκα θέσεων. Προσοχή, δεν είναι απαραίτητο ο arr να έχει το ίδιο μέγεθος με τον arr2, απλώς έτυχε στο παράδειγμα μας. Αυτό που πρέπει να προσέχουμε είναι ότι ο arr πρέπει να έχει το ίδιο μέγεθος με τον b και ο arr2 με τον c της διαδικασίας func1. Επίσης παρατηρείστε ότι το πλήθος τον ορισμάτων είναι ίσο σε αντιστοιχία ένα-προς-ένα. Το αρχείο με τα δεδομένα δοκιμής φαίνεται στην Εικόνα 9. Δεν γνωρίζουμε τις τιμές εξόδου και ζητάμε από τον NacVM να τις υπολογίσει. Τα αποτελέσματα των έξι σεναρίων φαίνονται παρακάτω:

```
00000001      0
00000002      0
00000003      0
00000004      0
00000005      0
00000006      0
```

Εικόνα 9. arrayargs_test_data.txt

```
*******NacVM RESULTS******

Nac File: [tests/arrayargs.nac]
Input Data File: [tests/arrayargs_test_data.txt]

1: Wrong[5]
2: Wrong[a]
3: Wrong[11]
4: Wrong[1c]
5: Wrong[29]
6: Wrong[3a]

-Elapsed time: 0.000 seconds.
```

Εικόνα 10. Αποτελέσματα εκτέλεσης σεναρίων για το arrayargs.nac.

Παρατηρείστε ότι ο χρόνος εκτέλεσης και για τα έξι σενάρια είναι μικρότερος από ένα millisecond. Το τρίτο παράδειγμα μας είναι το arraycalls.nac

```
globalvar s32 arr[5]={1,2,3,4,5},arr2[5]={0,0,0,0,0};

procedure arraycalls (in s32 n, out s32 s)
{
  localvar s32 g, result, i, t, t0;

S_1:
  g <= mov n;
  t <= ldc 0;
  S_2 <= jmpun;
S_2:
  (result, arr2) <= funcl (g, arr);
  S_3 <= jmpun;
S_3:
  i <= ldc 0;
  S_4 <= jmpun;
S_4:
  t0 <= load arr2, i;
  t <= add t, t0;
  i <= add i, 1;
  S_4, S_EXIT <= jmplt i, 5;
S_EXIT:
  s <= add t, result;
}

procedure funcl (in s32 in1, in s32 b[5], out s32 s, out s32 b2[5])
{
  localvar s32 a, i, t0, t1, t2;
  localvar s32 c[5]={0,0,0,0,0};

S_1:
  a <= mov in1;
  i <= ldc 0;
  S_2 <= jmpun;
S_2:
  t0 <= load b, i;
  c <= store t0, i;
  i <= add i, 1;
  S_2, S_3 <= jmplt i, 5;
S_3:
  i <= ldc 1;
  S_4 <= jmpun;
S_4:
  t0 <= load c, i;
  t1 <= shl a, i;
  t2 <= add t0, t1;
  c <= store t2, i;
  i <= add i, 1;
  S_4, S_5 <= jmplt i, 5;
S_5:
  i <= ldc 0;
  S_6 <= jmpun;
```

```
S_6:
  t0 <= load c, i;
  b2 <= store t0, i;
  i <= add i, 1;
  S_6, S_EXIT <= jmplt i, 5;
S_EXIT:
  s <= ldc 0;
}
```

Εικόνα 11. Κλήση διαδικασίας με in-out ορίσματα μονοδιάστατους πίνακες και πλήθος ορισμάτων μεγαλύτερο από ένα (arraycalls.nac).

```
0000001a      0
000000b2      0
0000003c      0
000000d4      0
0000005e      0
000000f6      0
000000aa      0
000000bb      0
000000cc      0
000000dd      0
00000011      0
000000bc      0
00000033      0
000000de      0
00000055      0
000000ff      0
000000a1      0
000000b4      0
000000c3      0
000000d9      0
```

Εικόνα 12. arraycalls_test_data.txt

Τα αποτελέσματα που παράγει ο NacVM για τα είκοσι σενάρια που φαίνονται στην Εικόνα 12 είναι τα παρακάτω:

```
**********NacVM RESULTS**********

Nac File: [tests/arraycalls.nac3]
Input Data File: [tests/arraycalls_test_data.txt]

1: Wrong[31b]
2: Wrong[14eb]
3: Wrong[717]
4: Wrong[18e7]
5: Wrong[b13]
6: Wrong[1ce3]
7: Wrong[13fb]
8: Wrong[15f9]
9: Wrong[17f7]
10: Wrong[19f5]
11: Wrong[20d]
12: Wrong[1617]
13: Wrong[609]
14: Wrong[1a13]
15: Wrong[a05]
16: Wrong[1df1]
17: Wrong[12ed]
18: Wrong[1527]
19: Wrong[16e9]
20: Wrong[197d]

-Elapsed time: 0.015 seconds.
```

Εικόνα 13. Αποτελέσματα εκτέλεσης σεναρίων για το arraycalls.nac.

Παρατηρείστε ότι ο χρόνος εκτέλεσης είναι αρκετά ικανοποιητικός (15 milliseconds) για την εκτέλεση των είκοσι παραπάνω σεναρίων από ένα διερμηνευτικό προσομοιωτή. Τα τρία επόμενα παραδείγματα που ακολουθούν, αφορούν την επίλυση μικρών υπολογιστικών προβλημάτων. Το ένα από αυτά (gss.nac) θα λέγαμε ότι είναι και απαιτητικό σε υπολογιστικούς πόρους. Ας δούμε λοιπόν το κώδικα του gss.nac και το πρόβλημα που αυτό επιλύει.

```
// Filename: gss.nac
// Purpose : N-address code (NAC) implementation that solves the problem
//           of finding the maximum partial table sum of a table.
// Author  : Angelos Kanatsos (C) 2011
// Date    : 05-Dec-2011

procedure gss (in s32 init1,in s32 init2,in s32 init3,in s32 init4, in
s32 mrn, out s32 maxsum)
{
       localvar s32 table[100000], r_num, pos, pos2, t, t2, t3, init[4];
       localvar s32 q, temp, maxhere, maxsofar, val, N[1]={100000};
B:     val <= div mrn, 2;
S:
       init <= store init1,0;
       init <= store init2,1;
       init <= store init3,2;
       init <= store init4,3;
       maxsofar <= ldc 0;
       maxhere <= ldc 0;
S_1:
       (t) <= xorshift2 (init);
       r_num <= rem t, mrn;        //range [0 - mrn]
       r_num <= sub r_num, val;    //range [-val - val]
       table <= store r_num, pos;
       pos <= add pos, 1;
       pos2 <= ldc 0;
       S_2,S_QUAN <= jmpne pos, N;
S_2:
       t2 <= load init, pos2;
       t3 <= add t, t2;
       init <= store t3, pos2;
       pos2 <= add pos2, 1;
       S_1,S_2 <= jmpeq pos2, 4;
S_QUAN:
       q <= load table, pos2;
       temp <= add maxhere, q;
       maxhere <= muxgt temp, 0, temp, 0;
       maxsofar <= muxgt maxhere, maxsofar, maxhere, maxsofar;
       pos2 <= add pos2,1;
       S_QUAN,S_EXIT <= jmpne pos2, N;
S_EXIT:
       maxsum <= mov maxsofar;
       pos <= ldc 0;
}

// Purpose : N-address code (NAC) implementation for a representative
//           "xorshift" RNG. Xorshift is a category of pseudorandom number
//            generators designed by George Marsaglia. It repeatedly uses
//            the transform of exclusive or on a number with a bit shifted
//            version of it. This specific algorithm has a period of
//            2^128-1 and passes the Diehard tests.
// Author  : Nikolaos Kavvadias (C) 2010, 2011
// Date    : 21-Jan-2011
// Revision: 0.3.0 (21/01/11)
```

```
S_1:
  x1 <= load init_values, 0;
  y1 <= load init_values, 1;
  z1 <= load init_values, 2;
  w1 <= load init_values, 3;
  t1 <= shl x1, 11;
  t  <= xor x1, t1;
  x1  <= mov y1;
  y1  <= mov z1;
  z1  <= mov w1;
  t2 <= shr t, 8;
  t3 <= xor t, t2;
  t4 <= shr w1, 19;
  t5 <= xor t4, t3;
  t6 <= xor w1, t5;
  w1  <= mov t6;
  init_values <= store x1, 0;
  init_values <= store y1, 1;
  init_values <= store z1, 2;
  init_values <= store w1, 3;
  rngval <= mov w1;
}
```

Εικόνα 14. Υπολογισμός μέγιστου αθροίσματος υποπίνακα ενός πίνακα 10.000 ακεραίων. Η τιμή για κάθε ακέραιο κυμαίνεται από -100 έως 100. Οι ακέραιοι αυτοί παράγονται με «τυχαίο» τρόπο από την xorshift2 και αρχικοποιούν τον πίνακα table πριν ξεκινήσει η διαδικασία.

Ο αλγόριθμος που έχουμε υλοποιήσει στο παράδειγμα της Εικόνας 14 είναι γραμμικής πολυπλοκότητας. Το gss.nac δέχεται πέντε ορίσματα εισόδου και παράγει μία έξοδο. Τα τέσσερα πρώτα χρησιμοποιούνται στην xorshift2 σαν feed για την παραγωγή σε κάθε σενάριο των 10.000 τυχαίων ακεραίων. Με αυτό τον τρόπο πετυχαίνουμε σε κάθε σενάριο την παραγωγή διαφορετικών τιμών για τον table. Το πέμπτο όρισμα εισόδου αφορά τον προσδιορισμό της μέγιστης και ελάχιστης τιμής που θα μπορεί να παίρνει ο κάθε τυχαίος ακέραιος στον πίνακα table. Για παράδειγμα αν στο σενάριο μας η mrn πάρει την τιμή 000000c9 = 201, τότε η μέγιστη τιμή του ακέραιου θα είναι το 100 και η ελάχιστη το -100. Ενώ αν πάρει την τιμή 00000191 = 401, τότε η μέγιστη θα είναι το 200 και η ελάχιστη το -200. Το παραπάνω πρόβλημα για να έχει νόημα επίλυσης θα πρέπει στον πίνακα να εμφανίζονται και αρνητικοί αριθμοί με πιθανότητα ίση με αυτή ενός θετικού. Και αυτό το πετυχαίνουμε παραπάνω. Τέλος το όρισμα εξόδου της gss.nac έχει σε κάθε σενάριο την τιμή του μέγιστου αθροίσματος υποπίνακα. Τα αποτελέσματα της εκτέλεσης φαίνονται στον παρακάτω πίνακα και παρατηρούμε ότι αν και το πρόβλημα είναι απαιτητικό, ο χρόνος εκτέλεσης ανήλθε μόλις στα 2,745 δευτερόλεπτα (Εικόνα 15, αριστερός πίνακας). Θα υποστηρίζαμε ότι είναι ένας καλός χρόνος για τον υπολογισμό πέντε τέτοιων σεναρίων. Τολμήσαμε να δοκιμάσουμε την εκτέλεση των ίδιων σεναρίων με πλήθος 100.000 ακεραίων αντί για 10.000 (Εικόνα 15, δεξιός πίνακας).

```
**********NacVM RESULTS**********
__________________________________
Nac File: [tests/gss.nac]
Input Data File:
[tests/gss_test_data.txt]


1: Wrong[1109]
2: Wrong[16f1]
3: Wrong[1ff0]
4: Wrong[10dd]
5: Wrong[1c16]

-Elapsed time: 2.745 seconds.
```

```
**********NacVM RESULTS**********
__________________________________
Nac File: [tests/gss.nac]
Input Data File:
[tests/gss_test_data.txt]


1: Wrong[350c]
2: Wrong[79cc]
3: Wrong[9177]
4: Wrong[270c]
5: Wrong[7b8f]

-Elapsed time: 125.720 seconds.
```

Εικόνα 15. Αποτελέσματα εκτέλεσης σεναρίων για το gss.nac.

Το πρόγραμμα coins.nac δέχεται σαν είσοδο το ποσό εκφρασμένο σε cents και παράγει σαν έξοδο το ελάχιστο πλήθος νομισμάτων ευρώ που απαιτούνται για την κάλυψη του. Ο αλγόριθμος που εφαρμόζεται είναι γνωστός ως άπληστη μέθοδος επιλογής.

```
// Filename: coins.nac
// Purpose : N-address code (NAC) implementation for the coins
computation
//           problem solver.
// Author  : Nikolaos Kavvadias (C) 2010
// Date    : 18-Oct-2010
// Revision: 0.3.0 (18/10/10)
//           Initial version.

procedure coins (in s32 n_eurocents, out s32 n_coins_used)
{
  localvar s32 i, t, n, n_items,times;
  localvar s32 C_euro[15], D_euro[15];

S_1:
  n <= mov n_eurocents;
  i <= ldc 0;
  t <= ldc 1;
  C_euro <= store t, i;
  i <= ldc 1;
  t <= ldc 2;
  C_euro <= store t, i;
  i <= ldc 2;
  t <= ldc 5;
  C_euro <= store t, i;
  i <= ldc 3;
  t <= ldc 10;
  C_euro <= store t, i;
  i <= ldc 4;
  t <= ldc 20;
```

```
  C_euro <= store t, i;
  i <= ldc 5;
  t <= ldc 50;
  C_euro <= store t, i;
  i <= ldc 6;
  t <= ldc 100;
  C_euro <= store t, i;
  i <= ldc 7;
  t <= ldc 200;
  C_euro <= store t, i;
  i <= ldc 8;
  t <= ldc 500;
  C_euro <= store t, i;
  i <= ldc 9;
  t <= ldc 1000;
  C_euro <= store t, i;
  i <= ldc 10;
  t <= ldc 2000;
  C_euro <= store t, i;
  i <= ldc 11;
  t <= ldc 5000;
  C_euro <= store t, i;
  i <= ldc 12;
  t <= ldc 10000;
  C_euro <= store t, i;
  i <= ldc 13;
  t <= ldc 20000;
  C_euro <= store t, i;
  i <= ldc 14;
  t <= ldc 50000;
  C_euro <= store t, i;
  S_2 <= jmpun;
S_2:
  print C_euro;
  times <= mov 15;
  (n_items, D_euro) <= evalcoins(times, n, C_euro);
  S_EXIT <= jmpun;
S_EXIT:
  n_coins_used <= mov n_items;
}

procedure evalcoins (in s32 n, in s32 amount, in s32 C[15], out s32
ncoins, out s32 D[15])
{
  localvar s32 find, choice, num;
  localvar s32 t0, t1, t2, t3, i, t;
  localvar s32 c_choice, d_num;
  localvar s32 TEMP[15];
S_1:
  find <= mov amount;
  num <= ldc 0;
  choice <= sub n, 1;
  i <= ldc 0;
  t <= ldc 0;
  S_2 <= jmpun;
S_2:
  TEMP <= store t, i;
  i <= add i, 1;
  S_2, S_3 <= jmplt i, 15;
```

```
S_3:
  c_choice <= load C, choice;
  t3 <= setle c_choice, find;
  S_4, S_5 <= jmpeq t3, 1;
S_4:
  d_num <= load TEMP, num;
  d_num <= add d_num, c_choice;
  TEMP <= store d_num, num;
  num <= add num, 1;
  find <= sub find, c_choice;
  S_6 <= jmpun;
S_5:
  choice <= sub choice, 1;
  S_6 <= jmpun;
S_6:
  t0 <= seteq choice, 0;
  t1 <= seteq find, 0;
  t2 <= and t0, t1;
  S_7, S_3 <= jmpeq t2, 1;
S_7:
  i <= ldc 0;
  S_8 <= jmpun;
S_8:
  d_num <= load TEMP, i;
  D <= store d_num, i;
  i <= add i, 1;
  S_8, S_EXIT <= jmplt i, 15;
S_EXIT:
  ncoins <= mov num;
}
```

Εικόνα 16. Υπολογισμός ελάχιστου αριθμού νομισμάτων ευρώ με την άπληστη μέθοδο.

Ένα αρκετά πολύπλοκο πρόγραμμα γραμμένο από τον Νικόλαο Καββαδία το οποίο θα προσομοιώσουμε με δέκα διαφορετικά σενάρια. Τα δύο αποτελέσματα τα υπολογίσαμε εμείς, τα υπόλοιπα οχτώ δεν τα γνωρίζουμε. Τρέχουμε το NacVM και παίρνουμε τα παρακάτω αποτελέσματα.

```
**********NacVM RESULTS**********
_____________________________
Nac File: [tests/coins.nac]
Input Data File:
[tests/coins_test_data.txt]

1: Correct
2: Wrong[6]
3: Wrong[4]
4: Wrong[9]
5: Correct
6: Wrong[4]
7: Wrong[9]
8: Wrong[4]
9: Wrong[5]
10: Wrong[8]

-Elapsed time: 0.000 seconds.
```

Εικόνα 17. Αποτελέσματα εκτέλεσης σεναρίων για το coins.nac.

Παρατηρούμε ότι ο χρόνος υπολογισμού και των δέκα σεναρίων ήταν κάτω από ένα millisecond. Αρκετά γρήγορη παρουσιάστηκε και εδώ η NAC μέσω του NacVM.

5. Συμπεράσματα

Όλα τα παραπάνω και κυρίως το Κεφάλαιο με τα παραδείγματα προσομοίωσης των προγραμμάτων σε γλώσσα NAC μας οδηγούν στο συμπέρασμα ότι η εικονική μηχανή NacVM που υλοποιήσαμε καταφέρνει να ικανοποιεί σε πολύ μεγάλο βαθμό τις απαιτήσεις της NAC. Λαμβάνοντας υπόψη ότι πρόκειται για ένα διερμηνευτικό προσομοιωτή, παρατηρήσαμε ότι σε αρκετά δύσκολα υπολογιστικά προβλήματα (π.χ. gss.nac) ο NacVM κατάφερε σε μικρό χρονικό διάστημα να προσομοιώσει επιτυχώς τον κώδικα NAC σε πολλά διαφορετικά σενάρια κάθε φορά.

Σίγουρα όμως υπάρχουν πολλά περιθώρια βελτίωσης του ως προς την επίδοση του αλλά και ως προς το βαθμό κάλυψης προσομοίωσης των δυνατοτήτων της NAC. Η επίδοση του θα μπορούσε να βελτιωθεί σε μια μελλοντική έκδοση του, αν η στοίβα δεν θα «κουβαλάει» σε κάθε κόμβο της το όνομα της διαδικασίας στην οποία ανήκει η κάθε πρόταση του προγράμματος. Επίσης να μην απαιτείται η δεύτερη σάρωση της στοίβας πριν την εκτέλεση της *fetch_execute_cycle()* της μηχανής, η οποία όπως είχαμε αναφέρει αποτελεί την καρδιά της. Από την άλλη ο βαθμός της προσομοίωσης θα μπορούσε να αυξηθεί και άλλο αν στο μέλλον αρθεί ο περιορισμός που δεν αφήνει να υπάρχει πρόταση κλήσης διαδικασίας από την τελευταία διαδικασία του πηγαίου προγράμματος NAC αν αυτή δεν είναι η γονική. Επιπλέον θα μπορούσε να υποστηρίζεται η κλήση κάποιας διαδικασίας χωρίς να υπάρχει υποχρεωτικά τουλάχιστον ένα όρισμα εισόδου και ένα εξόδου σε αυτή, όπως ισχύει τώρα. Ακόμα, σύμφωνα και με την αναφορά του Νικόλαου Καββαδία [1], θα μπορούσε κάποια στιγμή να υποστηρίζει μακροεντολές και περισσότερους τύπους αριθμητικών δεδομένων (π.χ. unsigned integer, floating point κ.α.).

Όσον αφορά τη γλώσσα NAC, προκύπτει από τα παραδείγματα του προηγούμενου Κεφαλαίου αλλά και από τον ορισμό της, ότι πρόκειται για μια ισχυρή μορφή ενδιάμεσης αναπαράστασης κώδικα. Λέγοντας ισχυρή, εννοούμε ότι με ένα σχετικά μικρό σύνολο τελεστών (48) καταφέρνει να επιλύσει προβλήματα με ελάχιστο αριθμό προτάσεων NAC σε σχέση με το πλήθος που θα απαιτούνταν σε άλλες μορφές ενδιάμεσης αναπαράστασης. Σε αυτό βοηθάει η δυναμική της φύση, καθώς η κάθε πρόταση δε περιορίζεται σε τρεις διευθύνσεις, αλλά σε άπειρες θεωρητικά (πρακτικά η διαθέσιμη μνήμη της φυσικής μηχανής που τρέχει ο NacVM). Επίσης το μέγεθος των όρων των τελεστών της δεν είναι καθορισμένο και μπορεί να φτάνει θεωρητικά

το άπειρο σε μία πρόταση (πρακτικά το INT_MAX). Τέλος τη δυναμική της φύση συμπληρώνει το γεγονός ότι επιτρέπει την κλήση διαδικασιών με άπειρο αριθμό ορισμάτων εισόδου και εξόδου, όπου το κάθε ένα από αυτά μπορεί να είναι ένας μονοδιάστατος πίνακας οποιουδήποτε μεγέθους. Η υποστήριξη κλήσεων διαδικασιών προσφέρει μεγάλη μείωση επαναλαμβανόμενου κώδικα στην ενδιάμεση αναπαράσταση αυξάνοντας την ισχύ της αναπαράστασης.

Άρα θα λέγαμε ότι η NAC ισορροπεί κομψά ανάμεσα στα δύο κριτήρια που αναφέραμε στην εισαγωγή της εργασίας, δηλαδή το πλήθος εντολών για την αναπαράσταση του κώδικα μιας γλώσσας υψηλού επιπέδου που επιλύει κάποιο πρόβλημα, σε σχέση με το πλήθος των τελεστών που έχει στη διάθεση της.

Βιβλιογραφία

[1] Kavvadias Nikolaos, (2011), *NAC (N-address code) programming language reference*, (http://www.nkavvadias.com/hercules/nac-refman.html, last visit 10/12/2011).

[2] Aho A.V., Sethi R., Ullman J.D., editors. *Compilers: Principles, Techniques, and Tools*. Pearson Education, 2001.

[3] Papaspyrou N., Skordalakis E., editors. *Compilers*. Symmetria Publications, 2002

[4] LLVM Project, (http://www.llvm.org/docs/LangRef.html, last visit 10/12/2011).

[5] GCC GIMPLE, the GNU Compiler Collection, (http://gcc.gnu.org/, last visit 10/12/2011).

[6] A C language family frontend for LLVM, (http://clang.llvm.org, last visit 09/12/2011)

[7] COINS Compiler Infrastructure, (http://www.coins-project.org/international/, last visit 09/10/2011)

[8] LANCE Retargetable C Compiler, (http://www.lancecompiler.com/, last visit 09/10/2011)

[9] An Introduction to Machine SUIF and Its Portable Libraries for Analysis and Optimization (http://www.eecs.harvard.edu/hube/software/nci/overview.html, last visit 06/12/2011)

[10] R. Leupers, O. Whalen, M. Hahenauer, T. Kogel, and P. Marwedel, ‘‘An executable intermediate representation for retargetable compilation and highlevel code optimization,’’ in *Proceedings of the Third International Workshop on Systems, Architectures, Modeling, and Simulation (SAMOS 2003)*, Samos, Greece, July 21-23 2003, pp. 120–125.

ΠΑΡΑΡΤΗΜΑ Α

Υλοποιήσεις των N-Address Εντολών της NAC

```
//mov
int _mov_(int *dst1, int *src1)
{
        if (src1 != NULL && dst1 != NULL)
        {
            *dst1 = *src1;
             return 1;
        }
        return -1;
}

//ldc
int _ldc_(int *dst1, int *src1)
{
        if (src1 != NULL)
        {
            *dst1 = *src1;
            return 1;
        }
        return -1;
}

//jmpun
instruction *_jmpun_(char *label, char *procname)
{
      proc_lab *proc_temp;
      label_address *la_temp;
      proc_temp = pl_head;

       while(proc_temp != NULL)
       {
           if(strcmp(proc_temp -> proc_name, procname) == 0)
           {
               la_temp = proc_temp -> la;
               while (la_temp  != NULL)
               {
                   if (strcmp(la_temp -> label_name, label) == 0)
                       return la_temp -> address;
                   la_temp = la_temp -> next;
               }
           }
           proc_temp = proc_temp -> next;
       }
       fprintf(stderr, "\nError: The label: [%s] does not exist. Unable to
                                                          jmpun.\n", label);
       exit(-1);
   }

//jmpeq
instruction *_jmpeq_(char *label1, char *label2, char *procname, int *src1,
int *src2)
{
     proc_lab *proc_temp;
     label_address *la_temp;
     proc_temp = pl_head;

     char *label_to_jump = NULL;
```

```
     if (*src1 == *src2)
        label_to_jump = label1;
     else
        label_to_jump = label2;

     while(proc_temp != NULL)
     {
         if(strcmp(proc_temp -> proc_name, procname) == 0)
         {
            la_temp = proc_temp -> la;
            while (la_temp  != NULL)
            {
                if (strcmp(la_temp -> label_name, label_to_jump) == 0)
                    return la_temp -> address;
                la_temp = la_temp -> next;
            }
         }
         proc_temp = proc_temp -> next;
       }
       fprintf(stderr, "\nError: The label: [%s] does not exist. Unable to
jmpeq.\n", label_to_jump);
       exit(-1);
}

//jmpne
instruction *_jmpne_(char *label1, char *label2, char *procname, int *src1,
int *src2)
{
    proc_lab *proc_temp;
    label_address *la_temp;
    proc_temp = pl_head;

    char *label_to_jump = NULL;

    if (*src1 != *src2)
        label_to_jump = label1;
    else
        label_to_jump = label2;

    while(proc_temp != NULL)
    {
       if(strcmp(proc_temp -> proc_name, procname) == 0)
       {
           la_temp = proc_temp -> la;
           while (la_temp  != NULL)
           {
               if (strcmp(la_temp -> label_name, label_to_jump) == 0)
                   return la_temp -> address;
               la_temp = la_temp -> next;
            }
       }
       proc_temp = proc_temp -> next;
    }
    fprintf(stderr, "\nError: The label: [%s] does not exist. Unable to
jmpne.\n", label_to_jump);
       exit(-1);
    }

//jmplt
instruction *_jmplt_(char *label1, char *label2, char *procname, int *src1,
int *src2)
{
    proc_lab *proc_temp;
    label_address *la_temp;
    proc_temp = pl_head;

    char *label_to_jump = NULL;
```

```
    if (*src1 < *src2)
        label_to_jump = label1;
    else
        label_to_jump = label2;

    while(proc_temp != NULL)
    {
       if(strcmp(proc_temp -> proc_name, procname) == 0)
       {
           la_temp = proc_temp -> la;
           while (la_temp  != NULL)
           {
               if (strcmp(la_temp -> label_name, label_to_jump) == 0)
                   return la_temp -> address;
               la_temp = la_temp -> next;
           }
       }
           proc_temp = proc_temp -> next;
    }
       fprintf(stderr, "\nError: The label: [%s] does not exist. Unable to
jmplt.\n", label_to_jump);
       exit(-1);
 }

//jmple
instruction *_jmple_(char *label1, char *label2, char *procname, int *src1,
int *src2)
{
    proc_lab *proc_temp;
    label_address *la_temp;
    proc_temp = pl_head;

    char *label_to_jump = NULL;

    if (*src1 <= *src2)
        label_to_jump = label1;
    else
        label_to_jump = label2;

    while(proc_temp != NULL)
    {
       if(strcmp(proc_temp -> proc_name, procname) == 0)
       {
           la_temp = proc_temp -> la;
           while (la_temp  != NULL)
           {
               if (strcmp(la_temp -> label_name, label_to_jump) == 0)
                   return la_temp -> address;
               la_temp = la_temp -> next;
           }
       }
       proc_temp = proc_temp -> next;
   }
   fprintf(stderr, "\nError: The label: [%s] does not exist. Unable to
jmple.\n", label_to_jump);
   exit(-1);
}

//jmpgt
instruction *_jmpgt_(char *label1, char *label2, char *procname, int *src1,
int *src2)
{
    proc_lab *proc_temp;
    label_address *la_temp;
    proc_temp = pl_head;

    char *label_to_jump = NULL;
```

```
    if (*src1 > *src2)
        label_to_jump = label1;
    else
        label_to_jump = label2;

    while(proc_temp != NULL)
    {
       if(strcmp(proc_temp -> proc_name, procname) == 0)
       {
           la_temp = proc_temp -> la;
           while (la_temp  != NULL)
           {
               if (strcmp(la_temp -> label_name, label_to_jump) == 0)
                   return la_temp -> address;
               la_temp = la_temp -> next;
            }
        }
        proc_temp = proc_temp -> next;
    }
    fprintf(stderr, "\nError: The label: [%s] does not exist. Unable to
jmpgt.\n", label_to_jump);
    exit(-1);
}

//jmpge
instruction *_jmpge_(char *label1, char *label2, char *procname, int *src1,
int *src2)
{
    proc_lab *proc_temp;
    label_address *la_temp;
    proc_temp = pl_head;

    char *label_to_jump = NULL;

    if (*src1 >= *src2)
        label_to_jump = label1;
    else
        label_to_jump = label2;

    while(proc_temp != NULL)
    {
       if(strcmp(proc_temp -> proc_name, procname) == 0)
       {
           la_temp = proc_temp -> la;
           while (la_temp  != NULL)
           {
               if (strcmp(la_temp -> label_name, label_to_jump) == 0)
                   return la_temp -> address;
               la_temp = la_temp -> next;
           }
        }
        proc_temp = proc_temp -> next;
    }
    fprintf(stderr, "\nError: The label: [%s] does not exist. Unable to
jmpge.\n", label_to_jump);
    exit(-1);
}

//and
int _and_(int *dst1, int *src1, int *src2)
{
   if (src1 != NULL && src2 != NULL)
   {
       *dst1 = *src1 & *src2;
       return 1;
   }
   return -1;
}
```

```
//ior
int _ior_(int *dst1, int *src1, int *src2)
{
    if (src1 != NULL && src2 != NULL)
    {
        *dst1 = *src1 | *src2;
        return 1;
    }
    return -1;
}

//xor
int _xor_(int *dst1, int *src1, int *src2)
{
    if (src1 != NULL && src2 != NULL)
    {
        *dst1 = *src1 ^ *src2;
        return 1;
    }
    return -1;
}

//nand
int _nand_(int *dst1, int *src1, int *src2)
{
    if (src1 != NULL && src2 != NULL)
    {
        *dst1 = ~(*src1 & *src2);
        return 1;
    }
    return -1;
}

//nor
int _nor_(int *dst1, int *src1, int *src2)
{
    if (src1 != NULL && src2 != NULL)
    {
        *dst1 = ~(*src1 | *src2);
        return 1;
    }
    return -1;
}

//xnor
int _xnor_(int *dst1, int *src1, int *src2)
{
    if (src1 != NULL && src2 != NULL)
    {
        *dst1 = ~(*src1 ^ *src2);
        return 1;
    }
    return -1;
}

//not
int _not_(int *dst1, int *src1)
{
     if (src1 != NULL)
     {
         *dst1 = ~*src1;
         return 1;
     }
     return -1;
}

//add
int _add_(int *dst1, int *src1, int *src2)
```

```
{
    if (src1 == NULL || src2 == NULL)
        return -1;
    *dst1 = *src1 + *src2;
    return 1;
}

//sub
int _sub_(int *dst1, int *src1, int *src2)
{
    if (src1 == NULL || src2 == NULL)
        return -1;

    *dst1 = *src1 - *src2;
    return 1;
}

//neg
int _neg_(int *dst1, int *src1)
{
     if (src1 != NULL)
     {
         *dst1 = -*src1;
         return 1;
     }
     return -1;
}

//muxeq
int _muxeq_(int *dst1, int *src1, int *src2, int *src3, int *src4)
{
    if (src1 != NULL && src2 != NULL && src3 != NULL && src4 != NULL)
    {
        if (*src1 == *src2)
            *dst1 = *src3;
        else
            *dst1 = *src4;

        return 1;
    }
    return -1;
}

//muxne
int _muxne_(int *dst1, int *src1, int *src2, int *src3, int *src4)
{
    if (src1 != NULL && src2 != NULL && src3 != NULL && src4 != NULL)
    {
        if (*src1 != *src2)
            *dst1 = *src3;
        else
            *dst1 = *src4;
        return 1;
    }
    return -1;
}

//muxlt
int _muxlt_(int *dst1, int *src1, int *src2, int *src3, int *src4)
{
    if (src1 != NULL && src2 != NULL && src3 != NULL && src4 != NULL)
    {
        if (*src1 < *src2)
            *dst1 = *src3;
        else
            *dst1 = *src4;
        return 1;
    }
```

```
    return -1;
}

//muxle
int _muxle_(int *dst1, int *src1, int *src2, int *src3, int *src4)
{
    if (src1 != NULL && src2 != NULL && src3 != NULL && src4 != NULL)
    {
        if (*src1 <= *src2)
            *dst1 = *src3;
        else
            *dst1 = *src4;
        return 1;
    }
    return -1;
}

//muxgt
int _muxgt_(int *dst1, int *src1, int *src2, int *src3, int *src4)
{
    if (src1 != NULL && src2 != NULL && src3 != NULL && src4 != NULL)
    {
        if (*src1 > *src2)
            *dst1 = *src3;
        else
            *dst1 = *src4;
        return 1;
    }
    return -1;
}

//muxge
int _muxge_(int *dst1, int *src1, int *src2, int *src3, int *src4)
{
    if (src1 != NULL && src2 != NULL && src3 != NULL && src4 != NULL)
    {
        if (*src1 >= *src2)
            *dst1 = *src3;
        else
            *dst1 = *src4;
        return 1;
    }
    return -1;
}

//seteq
int _seteq_(int *dst1, int *src1, int *src2)
{
     if (src1 == NULL || src2 == NULL)
        return -1;

     *dst1 = (*src1 == *src2);
     return 1;
}

//setne
int _setne_(int *dst1, int *src1, int *src2)
{
     if (src1 == NULL || src2 == NULL)
        return -1;

     *dst1 = (*src1 != *src2);
     return 1;
}

//setlt
int _setlt_(int *dst1, int *src1, int *src2)
{
```

```
     if (src1 == NULL || src2 == NULL)
        return -1;

     *dst1 = (*src1 < *src2);
     return 1;
}

//setle
int _setle_(int *dst1, int *src1, int *src2)
{
     if (src1 == NULL || src2 == NULL)
        return -1;

     *dst1 = (*src1 <= *src2);
     return 1;
}

//setgt
int _setgt_(int *dst1, int *src1, int *src2)
{
     if (src1 == NULL || src2 == NULL)
        return -1;

     *dst1 = (*src1 > *src2);
     return 1;
}

//setge
int _setge_(int *dst1, int *src1, int *src2)
{
     if (src1 == NULL || src2 == NULL)
        return -1;

     *dst1 = (*src1 >= *src2);
     return 1;
}

//abs
int _abs_(int *dst1, int *src1)
{
     if (src1 != NULL)
     {
          *dst1 = abs(*src1);
          return 1;
     }
     return -1;
 }

//max
int _max_(int *dst1, int *src1, int *src2)
{
    if (src1 != NULL && src2 != NULL)
    {
        if (*src1 >= *src2)
            *dst1 = *src1;
        else
            *dst1 = *src2;
        return 1;
    }
    return -1;
}

//min
int _min_(int *dst1,int *src1, int *src2)
{
    if (src1 != NULL && src2 != NULL)
    {
        if (*src1 >= *src2)
```

```
            *dst1 = *src2;
        else
            *dst1 = *src1;
        return 1;
    }
    return -1;
}

//shl
int _shl_(int *dst1, int *src1, int *src2)
{
    if (src1 != NULL && src2 != NULL)
    {
        if (*src2 > 0)
        {
            *dst1 = *src1 << *src2;
            return 1;
        }
        else
        {
            fprintf(stderr,"\nError: The amount of shifts must be over zero
            times.\n");
            exit(-1);
         }

     }
     return -1;
}

//shr
int _shr_(int *dst1, int *src1, int *src2)
{
    if (src1 != NULL && src2 != NULL)
    {
        if (*src2 > 0)
        {
             *dst1 = *src1 >> *src2;
             return 1;
        }
        else
        {
           fprintf(stderr,"\nError: The amount of shifts must be over zero
           times.\n");
           exit(-1);
        }

     }
     return -1;
}

//rotl
int _rotl_(int *dst1, int *src1, int *src2)
{
    if (src1 == NULL || src2 == NULL)
        return -1;

    if (*src2 > 0)
    {
         *src1 = (*src1 << *src2) | (*src1 >> (sizeof(*src1)*8 - *src2));
         *dst1 = *src1;
    }
    else
    {
       fprintf(stderr,"\nError: The amount of rotations must be over zero
       times.\n");
       exit(-1);
    }
```

```
    return 1;
}

//rotr
int _rotr_(int *dst1, int *src1, int *src2)
{
    if (src1 == NULL || src2 == NULL)
        return -1;

    if (*src2 > 0)
    {
         *src1 = (*src1 >> *src2) | (*src1 << (sizeof(*src1)*8 - *src2));
         *dst1 = *src1;
    }
    else
    {
        fprintf(stderr,"\nError: The amount of rotations must be over zero
        times.\n");
        exit(-1);
    }

    return 1;
}

//mul
int _mul_(int *dst1, int *src1, int *src2)
{
     if (src1 != NULL && src2 != NULL)
     {
         *dst1 = *src1 * *src2;
         return 1;
     }
     return -1;
}

//divrem
int _divrem_(int *dst1, int *dst2, int *src1, int *src2)
{
    if (src1 != NULL && src2 != NULL)
    {
        if (*src2 == 0)
        {
            fprintf(stderr,"\nError: Division by zero.\n");
            exit(-1);
        }
        *dst1 = *src1 / *src2;
        *dst2 = *src1 % *src2;
        return 1;
     }
     return -1;
}

//div
int _div_(int *dst1, int *src1, int *src2)
{
    if (src1 != NULL && src2 != NULL)
    {
        if (*src2 == 0)
        {
            fprintf(stderr,"\nError: Division by zero.\n");
            exit(-1);
        }
        *dst1 = *src1 / *src2;
        return 1;
    }
    return -1;
}
//rem
```

```
int _rem_(int *dst1, int *src1, int *src2)
{
    if (src1 != NULL && src2 != NULL)
    {
        if (*src2 == 0)
        {
            fprintf(stderr,"\nError: Division by zero.\n");
            exit(-1);
        }
        *dst1 = *src1 % *src2;
        return 1;
     }
     return -1;
 }

//zxt
int _zxt_(int *dst1, int *src1)
{
    if (src1 == NULL)
        return -1;

    *dst1 = (unsigned int) *src1;
    return 1;
}

//sxt
int _sxt_(int *dst1, int *src1)
{
    if (src1 == NULL)
        return -1;

    *dst1 = (signed int) *src1;
    return 1;
}

//trunc
int _trunc_(int *dst1, int *src1)
{
    if (src1 == NULL)
        return -1;

    *dst1 = *src1 & 4294967295;
    return 1;
}

//load
int _load_(int *dst1, char *procname, char *varname, int *src2)
{
    range_loc *ptr = range_head;
    globalvar_list *glptr;
    localvar_list *llptr;
    init_array *iptr;

    proc_inarg *inptr;
    proc_outarg *outptr;

    unsigned long ctr = 0;

    if (*src2 < 0)
    {
        fprintf(stderr,"\nError: The second argument of LOAD instruction
        must be positive integer.\n");
        exit(-1);
    }
    while (ptr != NULL)
    {
        if (strcmp (ptr -> procname, procname) == 0)
        {
```

```
llptr = ptr -> lv_list;
inptr = ptr -> pinarg;
outptr = ptr -> poutarg;

 while (llptr != NULL)
 {
    if (strcmp (llptr ->varname, varname) == 0)
    {
        iptr = llptr -> array;
        while (ctr < *src2)
        {
            iptr = iptr -> next;
            if (iptr == NULL)
            {
                fprintf(stderr,"\nError: Array [%s] out of
                bounds.\n", varname);
                exit(-1);
             }
             ctr++;
         }
         *dst1 = iptr->value;
         return 1;
     }
     llptr = llptr  -> next;
 }

 while (inptr != NULL)
 {
     if (strcmp (inptr ->varname, varname) == 0)
     {
         iptr = inptr -> array;
         while (ctr < *src2)
         {
             iptr = iptr -> next;
             if (iptr == NULL)
             {
                 fprintf(stderr,"\nError: Array [%s] out of
                 bounds.\n", varname);
                 exit(-1);
             }
             ctr++;
          }
          *dst1 = iptr->value;
          return 1;
       }
       inptr = inptr -> next;
  }

  while (outptr != NULL)
  {
     if (strcmp (outptr ->varname, varname) == 0)
     {
         iptr = outptr -> array;
         while (ctr < *src2)
         {
             iptr = iptr -> next;
             if (iptr == NULL)
             {
                 fprintf(stderr,"\nError: Array [%s] out of
                 bounds.\n", varname);
                 exit(-1);
             }
             ctr++;
          }
          *dst1 = iptr->value;
          return 1;
      }
     outptr = outptr -> next;
```

```
                }
            }
            ptr = ptr -> next;
        }

        glptr = ghead;
        while (glptr != NULL)
        {
           if (strcmp (glptr ->varname, varname) == 0)
           {
               iptr = glptr -> array;
               while (ctr < *src2)
               {
                   iptr = iptr -> next;
                   if (iptr == NULL)
                   {
                       fprintf(stderr,"\nError: Array [%s] out of bounds.\n",
                       varname);
                       exit(-1);
                   }
                   ctr++;
               }
               *dst1 = iptr->value;
               return 1;
           }
           glptr = glptr -> next;
        }
     fprintf(stderr, "\nError: Variable [%s] does not exist at procedure
     [%s] either as globalvar or localvar.\n", varname, procname);
     exit(-1);
  }

//store
int _store_(char *procname, char *varname, int *src1, int *src2)
{
    range_loc *ptr = range_head;
    globalvar_list *glptr;
    localvar_list *llptr;
    init_array *ilptr;
    init_array *iptr;

    proc_inarg *inptr;
    proc_outarg *outptr;

    unsigned long ctr = 0;

    if (*src2 < 0)
    {
        fprintf(stderr,"\nError: The second argument of STORE instruction
        must be positive integer.\n");
        exit(-1);
    }
    while (ptr != NULL)
    {
        if (strcmp (ptr -> procname, procname) == 0)
        {
            llptr = ptr -> lv_list;
            inptr = ptr -> pinarg;
            outptr = ptr -> poutarg;

            while (llptr != NULL)
            {
                if (strcmp (llptr ->varname, varname) == 0)
                {
                    ilptr = llptr -> array;
                    while (ctr < *src2)
                    {
                        ilptr = ilptr -> next;
```

```
                    if (ilptr == NULL)
                    {
                        fprintf(stderr,"\nError: Array [%s] out of
                        bounds.\n", varname);
                        exit(-1);
                    }
                    ctr++;
                }
                ilptr->value = *src1;
                return 1;
            }
            llptr = llptr  -> next;
        }
    }
    ptr = ptr -> next;
}

while (inptr != NULL)
{
     if (strcmp (inptr ->varname, varname) == 0)
     {
          ilptr = inptr -> array;
          while (ctr < *src2)
          {
              ilptr = ilptr -> next;
              if (ilptr == NULL)
              {
                  fprintf(stderr,"\nError: Array [%s] out of
                  bounds.\n", varname);
                  exit(-1);
               }
               ctr++;
          }
          ilptr->value = *src1;
          return 1;
      }
      inptr = inptr -> next;
 }

 while (outptr != NULL)
 {
     if (strcmp (outptr ->varname, varname) == 0)
     {
          ilptr = outptr -> array;
          while (ctr < *src2)
          {
               ilptr = ilptr -> next;
               if (ilptr == NULL)
               {
                   fprintf(stderr,"\nError: Array [%s] out of
                   bounds.\n", varname);
                   exit(-1);
               }
               ctr++;
           }
           ilptr->value = *src1;
           return 1;
       }
       outptr = outptr -> next;
  }

   glptr = ghead;
   while (glptr != NULL)
   {
       if (strcmp (glptr ->varname, varname) == 0)
       {
           iptr = glptr -> array;
           while (ctr < *src2)
```

```
                {
                    iptr = iptr -> next;
                    if (iptr == NULL)
                    {
                        fprintf(stderr,"\nError: Array [%s] out of
                        bounds.\n", varname);
                        exit(-1);
                    }
                    ctr++;
                }
                iptr->value = *src1;
                return 1;
            }
            glptr = glptr -> next;
          }
        fprintf(stderr, "\nError: Variable [%s] does not exist at procedure
        [%s] either as globalvar or localvar.\n", varname, procname);
        exit(-1);
 }

//print
int _print_(char *procname, char *varname)
{
    range_loc *ptr;
    ptr = range_head;
    globalvar_list *glptr;
    glptr = ghead;
    localvar_list *llptr;
    init_array *llarptr, *inptr,  *outptr, *init_gv;
    proc_inarg *ptr_list_in;
    proc_outarg *ptr_list_out;

    int tmp=0;

    while (ptr != NULL)
    {
        if (strcmp(ptr->procname,procname) == 0)
        {
            llptr = ptr -> lv_list;
            llarptr = llptr -> array;

            ptr_list_in = ptr -> pinarg;
            inptr = ptr_list_in -> array;
            ptr_list_out = ptr -> poutarg;
            outptr = ptr_list_out -> array;

            init_gv = glptr -> array;

            while (llptr != NULL)
            {
                if (strcmp(llptr->varname,varname) == 0)
                {
                    llarptr = llptr -> array;
                    fprintf(stdout,"%s:\n", llptr->varname);
                    tmp = llptr -> array_size;
                    while ( tmp > 0)
                    {
                        fprintf(stdout,"Hex:[%x] - Dec:[%d]\n",llarptr ->
                        value, llarptr -> value);
                        llarptr = llarptr -> next;
                        tmp--;
                    }
                    printf("\n");
                    return 1;
                }
                llptr = llptr -> next;
             }
               while (ptr_list_in != NULL)
```

```
            {
                 if (strcmp(ptr_list_in -> varname,varname) == 0)
                 {
                     inptr = ptr_list_in -> array;
                     fprintf(stdout,"%s:\n", ptr_list_in -> varname);
                     tmp = ptr_list_in -> array_size;
                     while (tmp > 0)
                     {
                         fprintf(stdout,"Hex:[%x] - Dec:[%d]\n",inptr ->
                         value, inptr -> value);
                         inptr = inptr -> next;
                         tmp--;
                     }
                     printf("\n");
                     return 1;
                 }
                 ptr_list_in = ptr_list_in -> next;
             }

             while (ptr_list_out != NULL)
             {
                 if (strcmp(ptr_list_out -> varname,varname) == 0)
                 {
                     outptr = ptr_list_out -> array;
                     fprintf(stdout,"%s:\n", ptr_list_out -> varname);
                     tmp = ptr_list_out -> array_size;
                     while (tmp > 0)
                     {
                         fprintf(stdout,"Hex:[%x] - Dec:[%d]\n",outptr –
                         >value, outptr->value);
                         outptr = outptr -> next;
                         tmp--;
                     }
                     printf("\n");
                     return 1;
                 }
                 ptr_list_out = ptr_list_out -> next;
             }
         }
         ptr = ptr -> next;
     }

   while (glptr != NULL)
   {
         if (strcmp(glptr -> varname,varname) == 0)
         {
             init_gv = glptr -> array;
             fprintf(stdout,"%s:\n", glptr -> varname);
             tmp = glptr -> array_size;
             while ( tmp > 0)
             {
                 fprintf(stdout,"Hex:[%x] - Dec:[%d]\n",init_gv -> value,
                 init_gv -> value);
                 init_gv = init_gv -> next;
                 tmp--;
              }
              printf("\n");
              return 1;
          }
          glptr = glptr -> next;
     }
     return -1;
}
```

ΠΑΡΑΡΤΗΜΑ Β

Στοίβα μηχανής, Πίνακας Συμβόλων, Βοηθητικές δομές και Χρήσιμες Συναρτήσεις του NacVM

```
/*
 * Filename: nacparse.c
 * Purpose : Top-level (driver) file of the "nacparse" program source code.
 * Author  : Angelos Kanatsos (C) 2011
 * Date    : 05-Sep-2011
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <time.h>

#define ALL 1
#define IN_OUT 2
#define ARRAY 1
#define ALL_GLOB 2
#define ALL_LOC 2

extern FILE *nacin;
extern unsigned long mass_proc_in_args;
extern unsigned long mass_proc_out_args;
FILE *nac_input_data;

char *NAC_FILE_NAME, *INPUT_DATA_FNAME;

clock_t start, end;

void nacparse(void);

static void print_usage()
{
  printf("\n");
  printf("* Usage:\n");
  printf("* nacparse [options] input.nac input.data\n");
  printf("* \n");
  printf("* Options:\n");
  printf("* \n");
  printf("*   -h:\n");
  printf("*         Print this help.\n");
  printf("* \n");
  printf("* For further information, please refer to the website:\n");
  printf("* http://www.nkavvadias.co.cc\n\n");
}

//main
int main(int argc, char **argv)
{
  int i;
  // Read input arguments
  if (argc != 3)
```

```
{
  print_usage();
  exit(1);
}

for (i=1; i < argc; i++)
{
      if (strcmp("-h", argv[i]) == 0)
      {
          print_usage();
      }
      else
      {
          if (argv[i][0] != '-')
          {
              if ((nacin = fopen(argv[i], "r")) == NULL)
              {
                  fprintf(stderr,"\nError: Can not open NAC file <%s>.\n",
                  argv[i]);
                  exit(-1);
              }
              else
              {
                  NAC_FILE_NAME = (char *)malloc(strlen(argv[i])*sizeof
                  (char));
                  strcpy(NAC_FILE_NAME,argv[i]);
              }
              i++;
              if ((nac_input_data = fopen(argv[i], "r")) == NULL)
              {
                  fprintf(stderr,"\nError: Can not open NAC TEST DATA file
                  <%s>.\n", argv[i]);
                  exit(-1);
              }
              else
              {
                  INPUT_DATA_FNAME = (char *)malloc(strlen(argv[i])
                  *sizeof(char));
                  strcpy(INPUT_DATA_FNAME,argv[i]);
              }
          }
      }
}
start = clock();

//Αρχικοποιήσεις δομών
initialize_stack(ALL);
initialize_globalvar(ALL_GLOB);
initialize_localvar(ALL_LOC);
initialize_call_list();

//Λεκτική, συντακτική και σημασιολογική ανάλυση
nacparse();
//Προσθέτουμε μία εικονική διαδικασία στο τέλος της στοίβας
//ώστε να επιτρέπονται κλήσεις διαδικασιών από τη τελευταία
//διαδικασία του αρχείου .nac
dummy_procedure();

//print_stack();
```

```
  //print_localvar_for_list();
  //print_globalvar();
  create_proc_list();
  //print_quick_access_list();
  read_data_from_file(nac_input_data);
  //print_file_data();


  fetch_execute_cycle();

  end = clock();

  printf("\n\n-Elapsed time: %.3f seconds.\n",(float)(end-start) /
  CLOCKS_PER_SEC);

  terminate();
  return 0;
}
```

```
/*
 * Filename: sym_tab_nac.h
 * Purpose : Top-level header file of the symbol table structure and more
             other useful functions.
 * Author  : Angelos Kanatsos (C) 2011
 * Date    : 05-Oct-2011
 */


#ifndef SYM_TAB_NAC_H
#define SYM_TAB_NAC_H

#include "stack_mac_nac.h"
#include <limits.h>

#define ARRAY 1
#define ALL_GLOB 2
#define ALL_LOC 2
#define HEAD_LOC 3

char *GLOBALVAR_NAME = NULL, *GLOBALVAR_TYPE = NULL;
char *LOCALVAR_NAME = NULL, *LOCALVAR_TYPE = NULL;
char *PROCEDURE_INOUT_TYPE = NULL;
extern char *NAC_FILE_NAME, *INPUT_DATA_FNAME;

unsigned long gv_rank = 1, gv_index = 0;
unsigned long lv_rank = 1, lv_index = 0;
unsigned long proc_in_rank = 1, proc_out_rank = 1, in_index = 0, out_index =
0, lv_list_rank = 1;

 struct array{
    int value;
    unsigned long index;

    struct array *next;
};
typedef struct array init_array;
init_array *arhead, *artail,*arlochead, *arloctail;

/**********GLOBALVAR************/
struct globalvar{
    char *varname;
    char *vartype;
    unsigned long array_size;
    init_array *array;
    unsigned long gv_rank;

    struct globalvar *next;
};
typedef struct globalvar globalvar_list;
globalvar_list *ghead, *gtail;

/**********LOCALVAR************/
struct localvar{
    char *varname;
    char *vartype;
    unsigned long array_size;
    init_array *array;
    unsigned long lv_rank;
```

```
    struct localvar *next;
};
typedef struct localvar localvar_list;
localvar_list *lhead, *ltail;

/*********SYMBOL TABLE*********/
struct proc_in_arguments {
    char *varname;
    char *vartype;
    unsigned long array_size;
    init_array *array;
    unsigned long in_rank;

    struct proc_in_arguments *next;
};
typedef struct proc_in_arguments proc_inarg;
proc_inarg *proc_inhead,*proc_intail;

struct proc_out_arguments {
    char *varname;
    char *vartype;
    unsigned long array_size;
    init_array *array;
    unsigned long out_rank;

    struct proc_out_arguments *next;
};
typedef struct proc_out_arguments proc_outarg;
proc_outarg *proc_outhead,*proc_outail;

struct range_localvar{
    char *procname;
    proc_inarg *pinarg;
    proc_outarg *poutarg;
    localvar_list *lv_list;
    unsigned long lv_list_rank;

    struct range_localvar *next;
};
typedef struct range_localvar range_loc;
range_loc *range_head, *range_tail;

/************FILE**************/
struct input_args{
    int in_arg_value;
    unsigned long in_arg_rank;
    struct input_args *next;
};
typedef struct input_args ina;
ina *in_args_head, *in_args_tail;

struct output_args{
    int out_arg_value;
    unsigned long out_arg_rank;
    struct output_args *next;
};
typedef struct output_args  outa;
outa *out_args_head, *out_args_tail;
```

```
struct input_data_from_file_to_nacvm {
    ina *in;
    outa *out;
    unsigned long number_of_lines;
    struct input_data_from_file_to_nacvm *next;
};
typedef struct input_data_from_file_to_nacvm file;
file *file_head = NULL, *file_tail = NULL;

// print_file_data
int print_file_data()
{
    file *tmp_file = file_head;
    ina *tmp_in;
    outa *tmp_out;

    printf("\n___________INPUT DATA___________\n\n   ");

    while(tmp_file -> next != NULL)
    {
        printf("\n%ld: ",tmp_file -> number_of_lines);

        tmp_in = tmp_file -> in;
        while(tmp_in != NULL)
        {
            printf("%x\t",tmp_in -> in_arg_value);
            tmp_in = tmp_in -> next;
        }

        tmp_out = tmp_file -> out;
        while(tmp_out != NULL)
        {
            printf("%x\t",tmp_out -> out_arg_value);
            tmp_out = tmp_out -> next;
        }

        tmp_file = tmp_file -> next;
    }
    printf("\n\n");
    return 0;
}

// pass_by_value_in
//Η πρώτη procedure είναι πάντα στο range_head!
//Στη πρώτη procedure περνάνε οι τιμές από το αρχείο.
int pass_by_value_in(ina *in)
{
    range_loc *ptr;
    ptr = range_head;
    ina *inaptr;
    inaptr = in;

    proc_inarg *inptr;
    inptr = ptr -> pinarg;

    while(inptr != NULL)
    {
        inptr -> array -> value = inaptr -> in_arg_value;
```

```
        inptr = inptr -> next;
        inaptr = inaptr -> next;
    }
    return 0;
}

// pass_by_value_out
int pass_by_value_out(outa *out, unsigned long lines)
{
    range_loc *ptr;
    ptr =range_head;
    outa *outaptr;
    outaptr = out;

    proc_outarg *outptr = ptr -> poutarg;

    while(outptr != NULL)
    {
        if (check_output_value(&outptr->array->value,&outaptr->
        out_arg_value,lines) != 0)
        {
            fprintf(stderr,"\nError: Failure to check the output value:
            [%s].\n", outptr->varname);
            exit(-1);
        }

        outptr = outptr -> next;
        outaptr = outaptr -> next;
    }
    return 0;
}

// checks_if_proc_exists
int checks_if_proc_exists(char *procname)
{
    range_loc *ptr;
    ptr = range_head;

    while(ptr != NULL)
    {
        if(strcmp(procname,ptr -> procname) == 0)
            return 1;

        ptr = ptr -> next;
    }
    fprintf(stderr,"\nError: Procedure [%s] does not exist at file [%s].\n",
    procname, (char *)NAC_FILE_NAME);
    exit(-1);
}

// check_type_size
void check_type_size(char *varname, char *size)
{
    if (LOCALVAR_TYPE != NULL)
    {
        if(strcmp(LOCALVAR_TYPE,"s32") != 0)
        {
            fprintf(stderr,"\nError: Localvar type:[%s] is not
            supported.\n\n",LOCALVAR_TYPE);
```

```
            exit(-1);
        }
    }
    else if (GLOBALVAR_TYPE != NULL)
    {
        if(strcmp(GLOBALVAR_TYPE,"s32") != 0)
        {
            fprintf(stderr,"\nError: Globalvar type:[%s] is not
            supported.\n\n",GLOBALVAR_TYPE);
            exit(-1);
        }
    }
    else
    {
        if(strcmp(PROCEDURE_INOUT_TYPE,"s32") != 0)
        {
            fprintf(stderr,"\nError: Type:[%s] is not
            supported.\n\n",PROCEDURE_INOUT_TYPE);
            exit(-1);
        }
    }
    if (atol(size) < 1)
    {
        fprintf(stderr,"\nError: The size of variable:[%s] can not be less
        than one.\n\n",varname);
        exit(-1);
    }
    if (atol(size) > INT_MAX)
    {
        fprintf(stderr,"\nError: The size of variable:[%s] can not be
        greater than INT_MAX = %d.\n\n",varname, INT_MAX);
        exit(-1);
    }
}

// initialize_globalvar
void initialize_globalvar(short what)
{
    arhead = artail = (init_array *)malloc(sizeof(init_array));
    arhead -> value = 0;
    arhead -> index = 0;
    arhead -> next = NULL;

    if (what == ALL_GLOB)
    {
        ghead = gtail = (globalvar_list *)malloc(sizeof(globalvar_list));
        ghead -> varname = NULL;
        ghead -> vartype = NULL;
        ghead -> array_size = 1;
        ghead -> array = NULL;
        ghead -> gv_rank = 0;
        ghead -> next = NULL;
    }

    if ((ghead == NULL) || (arhead == NULL))
    {
        fprintf(stderr,"Error: Memory Allocation: Can not initialize
        globalvar.");
        exit(-1);
```

```
    }
}

// initialize_localvar
void initialize_localvar(short what)
{
    arlochead = arloctail = (init_array *)malloc(sizeof(init_array));
    arlochead -> value = 0;
    arlochead -> index = 0;
    arlochead -> next = NULL;

    if (what == HEAD_LOC)
    {
        lhead = ltail = (localvar_list *)malloc(sizeof(localvar_list));
        lhead -> varname = NULL;
        lhead -> vartype = NULL;
        lhead -> array_size = 1;
        lhead -> array = NULL;
        lhead -> lv_rank = 0;
        lhead -> next = NULL;

        proc_inhead = proc_intail=(proc_inarg *)malloc(sizeof(proc_inarg));
        proc_inhead -> varname = NULL;
        proc_inhead -> vartype = NULL;
        proc_inhead -> array_size = 1;
        proc_inhead -> array = NULL;
        proc_inhead -> in_rank = 0;
        proc_inhead -> next = NULL;

        proc_outhead=proc_outail=(proc_outarg *)malloc(sizeof(proc_outarg));
        proc_outhead -> varname = NULL;
        proc_outhead -> vartype = NULL;
        proc_outhead -> array_size = 1;
        proc_outhead -> array = NULL;
        proc_outhead -> out_rank = 0;
        proc_outhead -> next = NULL;
    }
    else if (what == ALL_LOC)
    {
        lhead = ltail = (localvar_list *)malloc(sizeof(localvar_list));
        lhead -> varname = NULL;
        lhead -> vartype = NULL;
        lhead -> array_size = 1;
        lhead -> array = NULL;
        lhead -> lv_rank = 0;
        lhead -> next = NULL;

        range_head = range_tail = (range_loc *)malloc(sizeof(range_loc));
        range_head -> procname = NULL;
        range_head -> poutarg = NULL;
        range_head -> pinarg = NULL;
        range_head -> lv_list = NULL;
        range_head -> next = NULL;
        range_head -> lv_list_rank = 0;
    }

    if ((lhead == NULL) || (arlochead == NULL))
    {
        fprintf(stderr,"\nError: Memory Allocation: Can not initialize
```

```
        localvar.\n");
        exit(-1);
    }
}

// put_global_sym
globalvar_list *put_global_sym(char *name, char *size)
{
    globalvar_list *ptr;
    ptr = (globalvar_list *)malloc(sizeof(globalvar_list));
    ptr -> varname = (char *)malloc((strlen(name)+1) * sizeof(char));
    ptr -> vartype=(char *)malloc((strlen(GLOBALVAR_TYPE)+1)*sizeof(char));

    if ((ptr == NULL)||(ptr -> varname == NULL)||(ptr -> vartype == NULL))
    {
        fprintf(stderr,"\nError: Memory Allocation: At function
        put_global_sym().\n");
        exit(-1);
    }

    if (name != NULL)
    {
        strcpy(ptr -> varname, name);
    }

    if (GLOBALVAR_TYPE != NULL)
    {
        strcpy(ptr -> vartype, GLOBALVAR_TYPE);
    }

    if (size != NULL)
    {
        if(name != NULL)
            check_type_size(name,size);

        ptr -> array_size = atol(size);
        ptr -> gv_rank = gv_rank;
        ptr -> array = arhead;

        if ((ghead == gtail) && (gv_rank == 1))
            ghead = ptr;
        else
            gtail -> next = ptr;

        gtail = ptr;
        gtail -> next = NULL;

        initialize_globalvar(ARRAY);
        gv_index = 0;
        gv_rank++;
    }
    return gtail;
}

// put_gv_array_values
init_array *put_gv_array_values(char *value)
{
    init_array *ptr;
    ptr = (init_array *)malloc(sizeof(init_array));
```

```
    if (ptr == NULL)
    {
        fprintf(stderr,"\nError: Memory Allocation: At function
        put_gv_array_values()\n");
        exit(-1);
    }

    if (atol(value) < INT_MAX && atol(value) > INT_MIN)
        ptr -> value = atol(value);
    else if (atol(value) <= INT_MIN)
    {
        fprintf(stderr,"\nError: The value %s is less than INT_MIN =
        %d.\n", value, INT_MIN);
        exit(-1);
    }
    else
    {
        fprintf(stderr,"\nError: The value %s is greater than INT_MAX =
        %d.\n", value, INT_MAX);
        exit(-1);
    }

    ptr -> index = gv_index;

    if ((arhead == artail) && (gv_index == 0))
    {
        free(arhead);
        arhead = ptr;
    }
    else
        artail -> next = ptr;

    artail = ptr;
    artail -> next = NULL;

    gv_index++;
    return artail;
}

// put_proc_in_args
proc_inarg *put_proc_in_args(char *name, char *size)
{
    proc_inarg *ptr;
    ptr = (proc_inarg *)malloc(sizeof(proc_inarg));
    ptr -> varname = (char *)malloc((strlen(name)+1) * sizeof(char));
    ptr -> vartype = (char *)malloc((strlen(PROCEDURE_INOUT_TYPE)+1) *
    sizeof(char));

    if((ptr == NULL)||(ptr -> varname == NULL)||(ptr -> vartype == NULL))
    {
        fprintf(stderr,"\nError: Memory Allocation: At function
        put_proc_in_args().\n");
        exit(-1);
    }

    if (name != NULL)
    {
        strcpy(ptr -> varname, name);
```

```
    }

    if (PROCEDURE_INOUT_TYPE != NULL)
    {
        strcpy(ptr -> vartype, PROCEDURE_INOUT_TYPE);
    }

    if (size != NULL)
    {
        if(name != NULL)
            check_type_size(name,size);

        ptr -> array_size = atol(size);
        ptr -> in_rank = proc_in_rank;
        ptr -> array = arlochead;

        if ((proc_inhead == proc_intail) && (proc_in_rank == 1))
        {
            free(proc_inhead);
            proc_inhead = ptr;
        }
        else
            proc_intail -> next = ptr;

        proc_intail = ptr;
        proc_intail -> next = NULL;

        initialize_localvar(ARRAY);
        in_index = 0;
        proc_in_rank++;
    }
    return proc_intail;
}

// put_proc_out_args
proc_outarg *put_proc_out_args(char *name, char *size)
{
    proc_outarg *ptr;
    ptr = (proc_outarg *)malloc(sizeof(proc_outarg));
    ptr -> varname = (char *)malloc((strlen(name)+1) * sizeof(char));
    ptr -> vartype = (char *)malloc((strlen(PROCEDURE_INOUT_TYPE)+1) *
   sizeof(char));

    if ((ptr == NULL)||(ptr -> varname == NULL)||(ptr -> vartype == NULL))
    {
        fprintf(stderr,"\nError: Memory Allocation: At function
        put_proc_out_args().\n");
        exit(-1);
    }

    if (name != NULL)
    {
        strcpy(ptr -> varname, name);
    }

    if (PROCEDURE_INOUT_TYPE != NULL)
    {
        strcpy(ptr -> vartype, PROCEDURE_INOUT_TYPE);
    }
```

```
    if (size != NULL)
    {
       if(name != NULL)
           check_type_size(name,size);

        ptr -> array_size = atol(size);
        ptr -> out_rank = proc_out_rank;
        ptr -> array = arlochead;

        if ((proc_outhead == proc_outail) && (proc_out_rank == 1))
            proc_outhead = ptr;
        else
            proc_outail -> next = ptr;

        proc_outail = ptr;
        proc_outail -> next = NULL;

        initialize_localvar(ARRAY);
        out_index = 0;
        proc_out_rank++;
    }
    return proc_outail;
}

// put_local_sym
localvar_list *put_local_sym(char *name, char *size)
{
    localvar_list *ptr;
    ptr = (localvar_list *)malloc(sizeof(localvar_list));
    ptr -> varname = (char *)malloc((strlen(name)+1) * sizeof(char));
    ptr -> vartype = (char *)malloc((strlen(LOCALVAR_TYPE)+1) *
sizeof(char));

    if ((ptr == NULL)||(ptr -> varname == NULL)||(ptr -> vartype == NULL))
    {
        fprintf(stderr,"\nError: Memory Allocation: At function
        put_local_sym().\n");
        exit(-1);
    }

    if (name != NULL)
    {
        strcpy(ptr -> varname, name);
    }

    if (LOCALVAR_TYPE != NULL)
    {
        strcpy(ptr -> vartype, LOCALVAR_TYPE);
    }

    if (size != NULL)
    {
        if(name != NULL)
            check_type_size(name,size);

        ptr -> array_size = atol(size);
        ptr -> lv_rank = lv_rank;
        ptr -> array = arlochead;
```

```
        if ((lhead == ltail) && (lv_rank == 1))
            lhead = ptr;
        else
            ltail -> next = ptr;

        ltail = ptr;
        ltail -> next = NULL;

        initialize_localvar(ARRAY);
        lv_index = 0;
        lv_rank++;
    }
    return ltail;
}

// put_lv_array_values
init_array *put_lv_array_values(char *value, char what)
{
    init_array *ptr;
    ptr = (init_array *)malloc(sizeof(init_array));

    if (ptr == NULL)
    {
        fprintf(stderr,"\nError: Memory Allocation: At function
        put_lv_array_values().\n");
        exit(-1);
    }

    if (atol(value) < INT_MAX && atol(value) > INT_MIN)
        ptr -> value = atol(value);
    else if (atol(value) <= INT_MIN)
    {
         fprintf(stderr,"\nError: The value %s is less than INT_MIN =
         %d.\n", value, INT_MIN);
         exit(-1);
    }
    else
    {
        fprintf(stderr,"\nError: The value %s is greater than INT_MAX =
        %d.\n", value, INT_MAX);
        exit(-1);
    }

    switch (what)
    {
        case 'i':
            ptr -> index = in_index;
            break;
        case 'o':
            ptr -> index = out_index;
            break;
        case 'l':
            ptr -> index = lv_index;
            break;
    }

    if ((arlochead == arloctail) && (((lv_index == 0)&&(what == 'l')) ||
((in_index == 0)&&(what == 'i')) || ((out_index == 0)&&(what == 'o'))))
```

```
    {
        free(arlochead);
        arlochead = ptr;
    }
    else
        arloctail -> next = ptr;

    arloctail = ptr;
    arloctail -> next = NULL;

    switch (what)
    {
        case 'i':
            in_index++;
            break;
        case 'o':
            out_index++;
            break;
        case 'l':
            lv_index++;
            break;
    }
    return arloctail;
}

// create_lv_range_list
range_loc *create_lv_range_list(char *procedure_name, char *exist_lv)
{
    proc_in_rank = 1;
    proc_out_rank = 1;

    range_loc *ptr;
    ptr = (range_loc *)malloc(sizeof(range_loc));
    if (ptr == NULL)
    {
        fprintf(stderr,"\nError: Memory Allocation: At function
        create_lv_range_list().\n");
        exit(-1);
    }
    ptr -> procname = (char *)malloc((strlen(procedure_name)+1) *
    sizeof(char));
    strcpy(ptr -> procname, procedure_name);

    if (strcmp(exist_lv,"yes") == 0)
    {
        ptr -> lv_list = lhead;
        ptr -> lv_list_rank = lv_list_rank;
    }
    else
    {
        ptr -> lv_list = NULL;
        ptr -> lv_list_rank = 0;
    }
    ptr -> pinarg = proc_inhead;
    ptr -> poutarg = proc_outhead;

    if ((range_tail == range_head) && (lv_list_rank == 1))
    {
        free(range_head);
```

```
        range_head = ptr;
    }
    else
        range_tail -> next = ptr;

    range_tail = ptr;
    range_tail -> next = NULL;

    lv_list_rank++;
    lv_rank = 1;
    initialize_localvar(HEAD_LOC);
    return range_tail;
}

// analyze_token_array
long analyze_token_array(char *token, char kind_of_var)
{
    char *array_name = NULL, *array_size = NULL;
    char *temp_value = NULL;
    unsigned long temp_size = 0;
    int i = 0, j = 0;

    array_name = malloc((strlen(token)+1) * sizeof(char));
    array_size = malloc((strlen(token)+1) * sizeof(char));

    if ((array_name == NULL) || (array_size == NULL))
    {
        fprintf(stderr,"\nError: Memory Allocation: At function
        analyze_token_array().\n");
        exit(-1);
    }
    if ((strchr(token,'{') != NULL) && (strchr(token,'}') != NULL))
    {
        //copy array name
        while(token[j] != '[')
        {
            array_name[i++] = token[j++];
        }
        array_name[i] = '\0';

        //copy array size
        j++;
        i = 0;
        while(token[j] != ']')
        {
            array_size[i++] = token[j++];
        }
        array_size[i] = '\0';

        // Eat up '=' and '{'
        //Nac lex RE at line: 73
        j = j + 3;

        temp_value = (char *)malloc((strlen(token)+1) * sizeof(char));
        if ((array_name == NULL) || (array_size == NULL))
        {
            fprintf(stderr,"\nError: Memory Allocation: At function
            analyze_token_array().\n");
            exit(-1);
```

```
        }
        i = 0;
        while(token[j] != '}')
        {
            temp_value[i++] = token[j++];
            if ((token[j] == ',') || (token[j] == '}'))
            {
                temp_value[i] = '\0';
                if (kind_of_var == 'g')
                    put_gv_array_values(temp_value);
                else
                    put_lv_array_values(temp_value,kind_of_var);
                i = 0;
                if (token[j] == ',')
                    j++;
                else
                    break;
            }
        }
    }
    else if ((strchr(token,'[') != NULL) && (strchr(token,']') != NULL))
    {
        //copy array name
        while(token[j] != '[')
        {
            array_name[i++] = token[j++];
        }
        array_name[i] = '\0';

        //copy array size
        j++;
        i = 0;
        while(token[j] != ']')
        {
            array_size[i++] = token[j++];
        }
        array_size[i] = '\0';

        temp_size = atol(array_size);
        if (kind_of_var == 'g')
        {
            for(i=0;i<temp_size;i++)
            {
                put_gv_array_values("0");
            }
        }
        else if((kind_of_var=='l')||(kind_of_var=='i')||(kind_of_var== 'o'))
        {
            for(i=0;i<temp_size;i++)
            {
                put_lv_array_values("0",kind_of_var);
            }
        }
    }
    else
    {
        strcpy(array_name, token);
        strcpy(array_size, "1");
        if (kind_of_var == 'g')
```

```
            put_gv_array_values("0");
        else if((kind_of_var=='l')||(kind_of_var=='i')||(kind_of_var=='o'))
            put_lv_array_values("0",kind_of_var);
    }
    if (kind_of_var == 'g')
        put_global_sym(array_name, array_size);
    else if (kind_of_var == 'l')
        put_local_sym(array_name, array_size);
    else if (kind_of_var == 'i')
        put_proc_in_args(array_name, array_size);
    else if (kind_of_var == 'o')
        put_proc_out_args(array_name, array_size);
    return 0;
}

// value
//Η συνάρτηση αυτή επιστρέφει τη διεύθυνση της τιμής μιας μεταβλητής
//που αναζητούμε. Αρχικά ελέγχοι τις τοπικές μεταβλητές και στη συνέχεια
//τις καθολικές.
int *value(char *procname, char *varname, unsigned long index)
{
   range_loc *ptr_head;
   ptr_head = range_head;
   globalvar_list *ptr_head_gl;
   ptr_head_gl = ghead;

   localvar_list *ptr_list;
   proc_inarg *ptr_list_in;
   proc_outarg *ptr_list_out;

   unsigned long loops=0;

   while (ptr_head  != NULL)
   {
       ptr_list = ptr_head -> lv_list;
       ptr_list_in = ptr_head -> pinarg;
       ptr_list_out = ptr_head -> poutarg;

       if(strcmp(procname, ptr_head -> procname) == 0)
       {
           while(ptr_list !=NULL && ptr_list -> lv_rank != 0)
           {
               if(strcmp(varname, ptr_list -> varname) == 0)
               {
                   if(index <= ptr_list -> array_size)
                   {
                        while (loops < index)
                        {
                           ptr_list -> array =  ptr_list -> array -> next;
                           loops++;
                        }
                        return &ptr_list -> array -> value;
                   }
                   else
                   {
                       fprintf(stderr,"\nError: Index out of bounds: At
                       function value().\n");
                       exit(-1);
                   }
```

```
            }
            ptr_list = ptr_list -> next;
        }
        loops = 0;
        while(ptr_list_in != NULL && ptr_list_in -> in_rank != 0)
        {
            if(strcmp(varname, ptr_list_in -> varname) == 0)
            {
                if(index <= ptr_list_in -> array_size)
                {
                     while (loops < index)
                     {
                        ptr_list_in->array=ptr_list_in->array->next;
                        loops++;
                     }
                     return &ptr_list_in -> array -> value;
                }
                else
                {
                    fprintf(stderr,"\nError: Index out of bounds: At
                    function value().\n");
                    exit(-1);
                }
            }
            ptr_list_in = ptr_list_in -> next;
        }
        loops = 0;
        while(ptr_list_out != NULL && ptr_list_out != 0)
        {
            if(strcmp(varname, ptr_list_out -> varname) == 0)
            {
                if(index <= ptr_list_out -> array_size)
                {
                     while (loops < index)
                     {
                        ptr_list_out->array=ptr_list_out->array->next;
                        loops++;
                     }
                     return &ptr_list_out -> array -> value;
                }
                else
                {
                    fprintf(stderr,"\nError: Index out of bounds: At
                    function value().\n");
                    exit(-1);
                }
            }
            ptr_list_out = ptr_list_out -> next;
        }
    }
    ptr_head = ptr_head -> next;
}
loops = 0;
while (ptr_head_gl != NULL && ptr_head_gl -> gv_rank != 0)
{
     if(strcmp(varname, ptr_head_gl -> varname) == 0)
     {
         if(index <= ptr_head_gl -> array_size)
         {
```

```
                while (loops < index)
                {
                    ptr_head_gl -> array = ptr_head_gl -> array -> next;
                    loops++;
                }
                return &ptr_head_gl->array -> value;
            }
         }
        ptr_head_gl = ptr_head_gl -> next;
   }
   fprintf(stderr,"\nError: Variable [%s] is undefined.\n",varname);
   exit(-1);
}
```

```
/*
 * Filename: stack_mac_nac.h
 * Purpose : Top-level header file of the stack structure and more other
             useful functions on it.
 * Author  : Angelos Kanatsos (C) 2011
 * Date    : 05-Oct-2011
 */

#ifndef STACK_MAC_NAC_H
#define STACK_MAC_NAC_H

#include "sym_tab_nac.h"
#include "fetch_execute_cycle.h"
#include <stdlib.h>
#define ALL 1
#define IN_OUT 2
#define TRUE 1
#define FALSE 2

char *CUR_PROC_NAME = NULL;
char *CUR_LABEL_NAME = NULL;
char *OPERATION = NULL;
char *EMPTY = "Empty";
char PCALL = '0';

char *PREVIOUS_PROC_NAME = NULL;
unsigned short times = 0, procedure_changed = 0;
unsigned short first_time;

extern char *NAC_FILE_NAME, *INPUT_DATA_FNAME;
static unsigned int ctr = 0;

struct in_arguments {
    char *arg;
    unsigned int inrank;
    struct in_arguments *next;
};
typedef struct in_arguments inarg;
inarg *inhead,*intail;

struct out_arguments {
    char *arg;
    unsigned int outrank;
    struct out_arguments *next;
};
typedef struct out_arguments outarg;
outarg *outhead,*outail;

struct instructions {
    char *op;
    inarg *in;
    outarg *out;
    char *procname;  //Το όνομα της διαδικασίας στην οποία "ανήκει" η εντολή
    char *label;     //Η ετικέτα "μέσα" στην οποία βρίσκεται η εντολή
    char pcall;      //0 -> nac, 1 -> pcall
    unsigned int rank;
    struct instructions *next;
    struct instructions *previous;
};
```

```
typedef struct instructions instruction;
instruction *head ,*tail;

unsigned int outrank = 1, inrank = 1; //Πλήθος ορισμάτων εντολής (output
list) <= operation (input list)
unsigned int rank = 1;               //Πλήθος εντολών στη στοίβα μηχανής

struct stack_address_for_labels{
    char *label_name;
    instruction *address;

    struct stack_address_for_labels *next;
};
typedef struct stack_address_for_labels label_address;
label_address *la_head, *la_tail;

// Η δομή proc_lab αποτελεί μια "έξυπνη" δομή που λειτουργεί
//καταλυτικά στη γρήγορη και αποτελεσματική εκτέλεση των jmpzz instructions!
struct procedure_labels{
    char *proc_name;
    label_address *la;

    struct procedure_labels *next;
};
typedef struct procedure_labels proc_lab;
proc_lab *pl_head, *pl_tail;

//Η δομή proc_list περιέχει σε κάθε κόμβο πληροφορίες (διεύθυνση στοίβας,
//πλήθος ορισμάτων (in-out)) για κάθε procedure του προγράμματος. Έτσι όταν
//στο fetch_execute_cycle έχουμε κλήση διαδικασίας αυτή η δομή θα μας
//βοηθάει να "μεταβαίνουμε" και να "επιστρέφουμε" γρήγορα στη σωστή
//διεύθυνση της στοίβας.
struct proc_list{
    char *procname;
    instruction *address;
    unsigned short mass_of_in_args;
    unsigned short mass_of_out_args;

    struct proc_list *next;
};
typedef struct proc_list proc_list_acc;
proc_list_acc *proc_head, *proc_tail;

//Η δομή call_list περιέχει σε κάθε κόμβο τη διεύθυνση της στοίβας ΑΠΟ την
//οποία γίνεται η κλήση της διαδικασίας, αλλά και τη διεύθυνση της στοίβας
//στην οποία μεταβαίνει η ροή του προγράμματος. Επίσης περιέχει το όνομα της
//procedure από την οποία γίνεται η κλήση και το όνομα της procedure που
//καλείται
struct call_list{
    instruction *from_address;
    instruction *to_address;
    char *from_procedure;
    char *to_procedure;

    struct call_list *next;
    struct call_list *previous;
};
typedef struct call_list clist;
clist *clist_head,*clist_tail;
```

```
// initialize_call_list
void initialize_call_list()
{
    clist_head = clist_tail = (clist *)malloc(sizeof(clist));
    clist_head -> from_address = NULL;
    clist_head -> to_address = NULL;
    clist_head -> from_procedure = NULL;
    clist_head -> to_procedure = NULL;
    first_time = 1;

    if (clist_head == NULL)
    {
        fprintf(stderr,"\nError: Memory Allocation: Can not initialize
        call_list.\n");
        exit(-1);
    }
}

// do_call
int do_call(instruction *from_address, instruction *to_address, char *from,
char *to)
{
    clist *ptr;
    ptr = (clist *)malloc(sizeof(clist));
    if (ptr == NULL)
    {
            fprintf(stderr,"\nError: Memory Allocation: At function
            do_call()\n");
            exit(-1);
    }
    ptr -> from_procedure = (char *)malloc(strlen(from)*sizeof(char));
    ptr -> to_procedure = (char *)malloc(strlen(to)*sizeof(char));

    ptr -> from_address = from_address;
    ptr -> to_address = to_address;
    strcpy(ptr -> from_procedure,from);
    strcpy(ptr -> to_procedure,to);

    if (first_time == 1)
    {
        free(clist_head);
        clist_head = ptr;
        first_time = 0;
    }
    else
    {
        clist_tail -> next = ptr;
        ptr -> previous = clist_tail;
    }

    clist_tail = ptr;
    clist_tail -> next = NULL;
    return 0;
}

proc_list_acc *find_address(char *procname)
{
    proc_list_acc *ph;
    ph = proc_head;
```

```
    while (ph != NULL)
    {
        if(strcmp(ph->procname, procname) == 0)
            return ph;

        ph = ph -> next;
    }
    return NULL;
}

// read_data_from_file
int read_data_from_file(FILE *ptr_to_file)
{
   int counter, in_args_rank = 0, out_args_rank = 0;
   ina *ptr_in = NULL;
   outa *ptr_out = NULL;
   file *ptr_file = NULL;

   int file_value = 0;
   unsigned long lines = 1;

   while(!feof(ptr_to_file))
   {
        counter = 0;
        in_args_rank = 0;

        //Επειδή πάντα στη γονική περνάνε τα δεδομένα από το αρχείο
        while(counter < proc_head -> mass_of_in_args)
        {
            ptr_in = (ina *)malloc(sizeof(ina));
            if (ptr_in == NULL)
            {
                fprintf(stderr,"\nError: Memory Allocation: Can not read
                from file the input arguments.\n");
                exit(-1);
            }

            if(fscanf(ptr_to_file, "%x", &file_value) != 1)
            {
                break;
            }
            in_args_rank++;

            ptr_in -> in_arg_value = file_value;
            ptr_in -> in_arg_rank = in_args_rank;

            if (counter == 0 )
            {
                in_args_head = ptr_in;
            }
            else
                in_args_tail -> next = ptr_in;

            in_args_tail = ptr_in;
            in_args_tail -> next = NULL;
            counter++;
        }
       counter = 0;
```

```
        out_args_rank = 0;
        while(counter < proc_head -> mass_of_out_args)
        {
             ptr_out = (outa *)malloc(sizeof(outa));
             if (ptr_out == NULL)
             {
                 fprintf(stderr,"\nError: Memory Allocation: Can not read
                 from file the output arguments.\n");
                 exit(-1);
             }

             if(fscanf(ptr_to_file, "%x", &file_value) != 1)
             {
                 break;
             }
             out_args_rank++;

             ptr_out -> out_arg_value = file_value;
             ptr_out -> out_arg_rank = out_args_rank;

             if (counter == 0 )
             {
                 out_args_head = ptr_out;
             }
             else
                 out_args_tail -> next = ptr_out;

             out_args_tail = ptr_out;
             out_args_tail -> next = NULL;
             counter++;
         }

         ptr_file = (file *)malloc(sizeof(file));
         if (ptr_in == NULL)
         {
                 fprintf(stderr,"\nError: Memory Allocation: Can not read
                 from file the input arguments.\n");
                 exit(-1);
         }

         ptr_file -> in = in_args_head;
         ptr_file -> out = out_args_head;
         ptr_file -> number_of_lines = lines;

         if(file_head == NULL)
         {
             file_head = ptr_file;
         }
         else
             file_tail -> next = ptr_file;

         file_tail = ptr_file;
         file_tail -> next = NULL;

         lines++;
    }
     return 0;
}
```

```
// create_proc_list
int create_proc_list(void)
{
    instruction *ptr;
    proc_list_acc *temp;
    range_loc *temp_rl;
    proc_inarg *temp_ina;
    proc_outarg *temp_outa;

    ptr = head;
    proc_head = proc_tail = NULL;
    temp_rl = range_head;

    char *procedure;
    procedure = NULL;
    short flag = 0;
    unsigned int mass_of_in_args = 0, mass_of_out_args = 0;

    while(ptr != NULL)
    {
        temp = (proc_list_acc *)malloc(sizeof(proc_list_acc));
        if (temp == NULL)
        {
            fprintf(stderr,"\nError: Memory Allocation: At function
            create_proc_list()\n");
            exit(-1);
        }

        if(flag == 0 && ptr -> procname != NULL)
        {
            procedure = (char *)malloc(sizeof(char)*strlen(ptr ->
            procname));
            if (procedure == NULL)
            {
                fprintf(stderr,"\nError: Memory Allocation: At function
                create_proc_list()\n");
                exit(-1);
            }
            strcpy(procedure,ptr -> procname);

            temp_ina = temp_rl -> pinarg;
            while (temp_ina != NULL)
            {
                mass_of_in_args++;
                temp_ina = temp_ina -> next;
            }

            temp_outa = temp_rl -> poutarg;
            while (temp_outa != NULL)
            {
                mass_of_out_args++;
                temp_outa = temp_outa -> next;
            }

            temp -> procname = (char *)malloc(strlen(ptr -> procname) *
            sizeof(char));
            strcpy(temp -> procname, ptr -> procname);
            temp -> address = ptr;
            temp -> mass_of_in_args = mass_of_in_args;
```

```
        temp -> mass_of_out_args = mass_of_out_args;

        mass_of_in_args = mass_of_out_args = 0;
        temp_rl = temp_rl -> next;

         if (proc_head == NULL)
        {
                proc_head = temp;
        }
        else
        {
            proc_tail -> next = temp;
        }
        proc_tail = temp;
        proc_tail -> next = NULL;

        flag = 1;
    }

    if (strcmp(procedure,ptr->procname) != 0)
    {
        procedure = (char *)malloc(sizeof(char)*strlen(ptr->procname));
        strcpy(procedure,ptr->procname);

         temp_ina = temp_rl -> pinarg;
        while (temp_ina != NULL)
        {
            mass_of_in_args++;
            temp_ina = temp_ina -> next;
        }

        temp_outa = temp_rl -> poutarg;
        while (temp_outa != NULL)
        {
            mass_of_out_args++;
            temp_outa = temp_outa -> next;
        }

        temp -> procname = (char *)malloc(strlen(ptr -> procname) *
        sizeof(char));
        strcpy(temp -> procname, ptr -> procname);
        temp -> address = ptr;
        temp -> mass_of_in_args = mass_of_in_args;
        temp -> mass_of_out_args = mass_of_out_args;

        mass_of_in_args = mass_of_out_args = 0;
        temp_rl = temp_rl -> next;

         if (proc_head == NULL)
        {
                proc_head = temp;
        }
        else
        {
            proc_tail -> next = temp;
        }
        proc_tail = temp;
        proc_tail -> next = NULL;
    }
```

```
        ptr = ptr -> next;
    }
    return 0;
}

// free_nacsim
void free_nacsim(void)
{
    instruction *tmp;
    inarg *tmpin;
    outarg *tmpout;

    globalvar_list *tmpgv;
    init_array     *tmpar, *tmpinitar;

    localvar_list *tmpll;
    proc_inarg *tmpinarg;
    proc_outarg *tmpoutarg;
    range_loc *tmprl;

    proc_lab *pltmp;
    label_address *tmpla;

    proc_list_acc *platmp;
    clist *cltmp;

    file *f;
    ina *in;
    outa *out;

    while(file_head != NULL)
    {
        while (file_head -> in != NULL)
        {
            in = file_head -> in -> next;
            free(file_head -> in);
            file_head -> in = in;
        }
        while (file_head -> out != NULL)
        {
            out = file_head -> out -> next;
            free(file_head -> out);
            file_head -> out = out;
        }
        f = file_head -> next;
        free(file_head);
        file_head = f;
    }

    while (head != NULL)
    {
        while (head -> in != NULL)
        {
            tmpin = head -> in -> next;
            free(head -> in);
            head -> in = tmpin;
        }
        while (head -> out != NULL)
        {
```

```
            tmpout = head -> out -> next;
            free(head -> out);
            head -> out = tmpout;
        }
        tmp = head -> next;
        free(head);
        head = tmp;
    }

    while (proc_head != NULL)
    {
        platmp = proc_head -> next;
        free(proc_head);
        proc_head = platmp;
    }

    while (clist_head != NULL)
    {
        cltmp = clist_head -> next;
        free(clist_head);
        clist_head = cltmp;
    }

    while (ghead != NULL)
    {
        while (ghead -> array != NULL)
        {
            tmpar = ghead -> array -> next;
            free(ghead -> array);
            ghead -> array = tmpar;
        }
        tmpgv = ghead -> next;
        free(ghead);
        ghead = tmpgv;
    }

    while (pl_head != NULL)
    {
        while (pl_head -> la != NULL)
        {
            tmpla = pl_head -> la -> next;
            free(pl_head -> la);
            pl_head -> la = tmpla;
        }
        pltmp = pl_head -> next;
        free(pl_head);
        pl_head = pltmp;
    }

    while (range_head != NULL)
    {
        while (range_head -> lv_list != NULL)
        {
            while (range_head -> lv_list -> array != NULL)
            {
                tmpinitar = range_head -> lv_list -> array-> next;
                free(range_head -> lv_list -> array);
                range_head -> lv_list -> array = tmpinitar;
            }
```

```
            tmpll = range_head -> lv_list -> next;
            free(range_head -> lv_list);
            range_head -> lv_list = tmpll;
        }
        while (range_head -> pinarg != NULL)
        {
            while (range_head -> pinarg -> array != NULL)
            {
                tmpinitar = range_head -> pinarg -> array-> next;
                free(range_head -> pinarg -> array);
                range_head -> pinarg -> array = tmpinitar;
            }
            tmpinarg = range_head -> pinarg -> next;
            free(range_head -> pinarg);
            range_head -> pinarg = tmpinarg;
        }
        while (range_head -> poutarg != NULL)
        {
            while (range_head -> poutarg -> array != NULL)
            {
                tmpinitar = range_head -> poutarg -> array-> next;
                free(range_head -> poutarg -> array);
                range_head -> poutarg -> array = tmpinitar;
            }
            tmpoutarg = range_head -> poutarg -> next;
            free(range_head -> poutarg);
            range_head -> poutarg = tmpoutarg;
        }
        tmprl = range_head -> next;
        free(range_head);
        range_head = tmprl;
    }
}

// terminate
void terminate(void)
{
    fprintf(stdout,"\n\nΠιέστε το Enter  για να κλείσει η εφαρμογή...\n");
    if(getchar() == '\r\n')
    {
        free_nacsim();
        exit(1);
    }
}

// initialize_stack
void initialize_stack(short what)
{
    intail = inhead = ((inarg *)malloc(sizeof(inarg)));
    inhead -> arg = NULL;
    inhead -> inrank = 0;
    inhead -> next = NULL;

    outail = outhead = ((outarg *)malloc(sizeof(outarg)));
    outhead -> arg = NULL;
    outhead -> outrank = 0;
    outhead -> next = NULL;

    inrank = 1;
```

```
    outrank = 1;

    if (what == ALL)
    {
        tail = head = ((instruction *)malloc(sizeof(instruction)));
        head -> op = NULL;
        head -> in = NULL;
        head -> out = NULL;
        head -> procname = NULL;
        head -> label = NULL;
        head -> pcall = '0';
        head -> rank = 0;

        rank = 1;
    }

    if (((inhead == NULL) && (outhead == NULL)) || (head == NULL))
    {
        fprintf(stderr,"Error: Memory Allocation: Can not initialize
        stack.");
        terminate();
    }
}

// gen_inarg_list
inarg *gen_inarg_list(char *arg)
{
    inarg *ptr;
    ptr = (inarg *)malloc(sizeof(inarg));
    ptr -> arg = (char *)malloc(strlen(arg)+1);
    if((ptr == NULL) || (ptr -> arg == NULL))
    {
        fprintf(stderr,"Error: Memory Allocation: At function
        gen_inarg_list().");
        terminate();
    }
    strcpy(ptr -> arg, arg);
    ptr -> inrank = inrank;
    ptr -> next = NULL;

    if ((inhead == intail) && (inrank == 1))
    {
        free(inhead);
        inhead = ptr;
    }
    else
        intail -> next = ptr;

    intail = ptr;
    intail -> next = NULL;

    inrank++;
    return intail;
}

// gen_outarg_list
outarg *gen_outarg_list(char *arg)
{
    outarg *ptr;
```

```
    ptr = (outarg *)malloc(sizeof(outarg));
    ptr -> arg = (char *)malloc(strlen(arg)+1);
    if((ptr == NULL) || (ptr -> arg == NULL))
    {
        fprintf(stderr,"Error: Memory Allocation: At function
        gen_outarg_list().");
        terminate();
    }
    strcpy(ptr->arg,arg);
    ptr -> outrank = outrank;

    if ((outhead == outail) && (outrank == 1))
    {
        free(outhead);
        outhead = ptr;
    }
    else
        outail -> next = ptr;

    outail = ptr;
    outail -> next = NULL;

    outrank++;
    return outail;
}

// gen_stack
instruction *gen_stack(void)
{
    instruction *ptr;
    ptr = (instruction *)malloc(sizeof(instruction));
    if(ptr == NULL)
    {
        fprintf(stderr,"Error: Memory Allocation: At function
        gen_stack().");
        terminate();
    }

    if (OPERATION != NULL)
    {
        ptr -> op = (char *)malloc(sizeof(OPERATION));
        if(ptr -> op == NULL)
        {
            fprintf(stderr,"Error: Memory Allocation: At function
            gen_stack().");
            terminate();
        }
        if (PCALL == '0')
        {
            if (check_nac_statement(OPERATION) == 1)
                strcpy(ptr -> op, OPERATION);
            else if (check_nac_statement(OPERATION) == 2)
            {
                fprintf(stderr,"\nError: This version does not support
                operation: [%s]\n",OPERATION);
                exit(-1);
            }
            else
            {
```

```
            fprintf(stderr,"\nError: Unknown operation:
            [%s]\n",OPERATION);
            exit(-1);
        }
    }
    else
        strcpy(ptr -> op, OPERATION);
}
else
{
    ptr -> op = (char *)malloc(sizeof(EMPTY));
    strcpy(ptr -> op, EMPTY);
}
ptr -> in = inhead;
ptr -> out = outhead;

if (CUR_PROC_NAME != NULL)
{
    ptr -> procname = (char *)malloc(sizeof(CUR_PROC_NAME));
    if(ptr -> procname == NULL)
    {
        fprintf(stderr,"Error: Memory Allocation: At function
        gen_stack().");
        terminate();
    }
    strcpy(ptr -> procname, CUR_PROC_NAME);
}
else
{
    ptr -> procname = (char *)malloc(sizeof(EMPTY));
    strcpy(ptr -> procname, EMPTY);
}

if (CUR_LABEL_NAME != NULL)
{
    ptr -> label = (char *)malloc(sizeof(CUR_LABEL_NAME));
    if(ptr -> label == NULL)
    {
        fprintf(stderr,"Error: Memory Allocation: At function
        gen_stack().");
        terminate();
    }
    strcpy(ptr -> label, CUR_LABEL_NAME);
}
else
{
    ptr -> label = (char *)malloc(sizeof(EMPTY));
    strcpy(ptr -> label, EMPTY);
}
ptr -> pcall = PCALL;
ptr -> rank = rank;

if ((head == tail) && (rank == 1))
    head = ptr;
else
{
    tail -> next = ptr;
    ptr -> previous = tail;
}
```

```
    tail = ptr;
    tail -> next = NULL;

    if (CUR_PROC_NAME != NULL && CUR_LABEL_NAME != NULL)
            gen_list_acc_stack_labels(strdup(CUR_PROC_NAME),
    strdup(CUR_LABEL_NAME), ptr);

    initialize_stack(IN_OUT);
    PCALL = '0';
    rank++;
    return tail;
}

// gen_list_acc_stack_labels
int gen_list_acc_stack_labels(char *procname, char *labelname, instruction
*address)
{
    proc_lab *ptr;
    label_address *ptr2;
    label_address *la_tmp;

    if (times == 0)
    {
        PREVIOUS_PROC_NAME = (char *)malloc((strlen(procname) +1)*sizeof
        (char));
        strcpy(PREVIOUS_PROC_NAME, procname);
        times = 1;
    }
    else
    {
        if (strcmp(PREVIOUS_PROC_NAME, procname) == 0)
        {
            la_tmp = la_head;
            while (la_tmp != NULL)
            {
                if (strcmp(la_tmp -> label_name, labelname) == 0)
                    return 0;
                la_tmp = la_tmp -> next;
            }
            procedure_changed = 0;
        }
        else
        {
             strcpy(PREVIOUS_PROC_NAME, procname);
             procedure_changed = 1;
        }
    }
    ptr = (proc_lab *)malloc(sizeof(proc_lab));
    ptr2 = (label_address *)malloc(sizeof(label_address));
    ptr -> proc_name = (char *)malloc((strlen(procname)+1) * sizeof(char));
    ptr2 -> label_name=(char *)malloc((strlen(labelname)+1)*sizeof (char));

    if((ptr==NULL)||(ptr->proc_name==NULL)||(ptr2->label_name==NULL)||(ptr2
                                                                    ==NULL))
    {
        fprintf(stderr,"\nError: Memory Allocation: At function
        gen_list_acc_stack_labels()\n\n");
        exit(-1);
    }
```

```
    if (procname != NULL && labelname != NULL)
    {
        strcpy(ptr -> proc_name, procname);
        strcpy(ptr2 -> label_name, labelname);
        ptr2 -> address = address;

        if (ctr == 0)
        {
            pl_head = ptr;
            pl_tail = ptr;

            la_head = ptr2;
            la_tail = ptr2;

            pl_head -> la = la_head;

            la_head -> next = NULL;
            pl_head -> next = NULL;

            ctr = 1;
        }

        if (procedure_changed == 0)
        {
            la_tail  -> next = ptr2;
            la_tail = ptr2;
            la_tail -> next = NULL;

        }

         if (procedure_changed == 1)
         {
             pl_tail -> next = ptr;
             pl_tail = ptr;
             pl_tail -> next = NULL;

             la_head = ptr2;
             la_tail = ptr2;

             pl_tail -> la = la_head;;
             la_head -> next = NULL;
         }
    }
    return 0;
}

// copy_proc_in_args
//Η συνάρτηση αυτή χρησιμοποιείται στις κλήσεις διαδικασιών για την
//αντιγραφή των τιμών των input τιμών (PASS BY VALUE!!!)
int copy_proc_in_args(char *from_proc, char *to_proc, inarg *stack_in_args)
{
  range_loc *rlptr;
  rlptr = range_head;

  proc_inarg *from_inptr = NULL, *from_inptr_help = NULL, *to_inptr = NULL,
  *to_inptr_help = NULL;
  localvar_list *from_lvptr = NULL, *from_lvptr_help = NULL;

  inarg *ptr_inarg;
```

```
ptr_inarg = stack_in_args;

globalvar_list *glptr;
glptr = ghead;

short found = 0;
unsigned long counter = 0;

init_array *tmp_to=NULL, *tmp_from=NULL;

if (strcmp(from_proc,to_proc) != 0)
{
    while(rlptr != NULL)
    {
        if (strcmp(from_proc,rlptr -> procname) == 0)
        {
            from_lvptr_help = from_lvptr = rlptr -> lv_list;
            from_inptr_help = from_inptr = rlptr -> pinarg;
         }

         if (strcmp(to_proc,rlptr -> procname) == 0)
         {
              to_inptr_help = to_inptr = rlptr -> pinarg;
         }
         rlptr = rlptr -> next;
    }

    if (to_inptr == NULL)
    {
         fprintf(stderr,"\nError: Procedure [%s] does not exist.\n",
         to_proc);
         exit(-1);
    }

    while (to_inptr -> next != NULL)
    {
        to_inptr = to_inptr -> next;
    }
    while (ptr_inarg -> next != NULL)
    {
        ptr_inarg = ptr_inarg -> next;
    }

    if(to_inptr -> in_rank != ptr_inarg -> inrank)
    {
        fprintf(stderr,"\nError: Different mass of input arguments at
        call: [%s].\n\n", to_proc);
        exit(-1);
    }

    to_inptr = to_inptr_help;
    ptr_inarg = stack_in_args;
}
else
{
    fprintf(stderr,"\nError: Nac Virtual Machine does not support
    Recursion.\n");
    exit(-1);
}
```

```
while(ptr_inarg != NULL)
{
    from_lvptr = from_lvptr_help;
    while (from_lvptr != NULL)
    {
       if(strcmp(from_lvptr -> varname,ptr_inarg -> arg) == 0)
       {
           if(from_lvptr -> array_size == to_inptr -> array_size)
           {
               found = 1;
               tmp_to = to_inptr -> array;
               tmp_from = from_lvptr -> array;
               while(1)
               {
                   tmp_to -> index = tmp_from -> index;
                   tmp_to -> value = tmp_from -> value;
                   tmp_from = tmp_from -> next;
                   tmp_to =  tmp_to -> next;
                   counter++;
                   if (counter == from_lvptr->array_size)
                   {
                       counter = 0;
                       break;
                   }
                }
                to_inptr = to_inptr -> next;
             }
             else
             {
                fprintf(stderr,"\nError: Variables [%s] and [%s] have
                different array size at call: [%s].\n",from_lvptr ->
                varname,to_inptr -> varname,to_proc);
                exit(-1);
             }
          }
          from_lvptr = from_lvptr -> next;
    }

    //Αυξάνουμε τη ταχύτητα του NacVM.
    //Δε χρειάζεται να ελέγξει τα παρακάτω.
    if(found == 1)
    {
        ptr_inarg = ptr_inarg -> next;
        found = 0;
        continue;
    }

    from_inptr = from_inptr_help;
    while (from_inptr != NULL)
    {
       if(strcmp(from_inptr->varname,ptr_inarg->arg) == 0)
       {
           if(from_inptr->array_size == to_inptr->array_size)
           {
               found = 1;
               tmp_to = to_inptr -> array;
               tmp_from = from_inptr -> array;
               while(1)
               {
```

```
                tmp_to -> index = tmp_from -> index;
                tmp_to -> value = tmp_from -> value;
                tmp_from = tmp_from -> next;
                tmp_to = tmp_to -> next;
                counter++;
                if (counter == from_inptr->array_size)
                {
                   counter = 0;
                   break;
                }
            }
            to_inptr = to_inptr -> next;
        }
        else
        {
            fprintf(stderr,"\nError: Variables [%s] and [%s] have
            different array size at call: [%s].\n",from_inptr->varname,
            to_inptr-> varname,to_proc);
            exit(-1);
        }
    }
    from_inptr = from_inptr -> next;
}

//Αυξάνουμε τη ταχύτητα του NacVM.
//Δε χρειάζεται να ελέγξει τα παρακάτω.
if(found == 1)
{
    ptr_inarg = ptr_inarg -> next;
    found = 0;
    continue;
}

glptr = ghead;
while (glptr != NULL)
{
    if(strcmp(glptr->varname, ptr_inarg->arg) == 0)
    {
        found = 1;
        if (glptr->array_size == to_inptr->array_size)
        {
            tmp_to = to_inptr -> array;
            tmp_from = glptr -> array;
            while(1)
            {
                tmp_to -> index = tmp_from -> index;
                tmp_to -> value = tmp_from -> value;

                tmp_from = tmp_from->next;
                tmp_to = tmp_to -> next;
                counter++;
                if (counter == glptr->array_size)
                {
                   counter = 0;
                   break;
                }
            }
            to_inptr = to_inptr -> next;
        }
```

```
            else
            {
                fprintf(stderr,"\nError: Variables [%s] and [%s] have
                different array size at call: [%s].\n",glptr->varname,
                to_inptr->varname,to_proc);
                exit(-1);
            }
        }
        glptr = glptr -> next;
    }

    if(found == 0)
    {
        fprintf(stderr,"\nError: Variable [%s] at procedure [%s] has not
        defined.\n",ptr_inarg->arg,from_proc);
        exit(-1);
    }
    else
        found = 0;

    ptr_inarg = ptr_inarg -> next;
  }
  return 1;
 }

// copy_proc_out_args
int copy_proc_out_args(char *from_proc, char *to_proc,  outarg
*stack_out_args)
{
  range_loc *rlptr;
  rlptr = range_head;

  proc_outarg *from_outptr = NULL, *from_outptr_help = NULL, *to_outptr =
  NULL, *to_outptr_help = NULL;
  localvar_list *to_lvptr = NULL, *to_lvptr_help = NULL;

  outarg *ptr_outarg;
  ptr_outarg = stack_out_args;

  globalvar_list *glptr;
  glptr = ghead;

  short found = 0;
  unsigned long counter = 0;

  init_array *tmp_to=NULL,*tmp_from=NULL;

  if (strcmp(from_proc,to_proc) != 0)
  {
      while(rlptr != NULL)
      {
          if (strcmp(from_proc,rlptr -> procname) == 0)
          {
              from_outptr_help = from_outptr = rlptr -> poutarg;
          }

          if (strcmp(to_proc,rlptr -> procname) == 0)
          {
                to_lvptr_help = to_lvptr = rlptr -> lv_list;
```

```
                to_outptr_help = to_outptr = rlptr -> poutarg;
            }
            rlptr = rlptr -> next;
        }

        while (ptr_outarg -> next != NULL)
        {
            ptr_outarg = ptr_outarg -> next;
        }
        while (from_outptr -> next != NULL)
        {
            from_outptr = from_outptr -> next;
        }

        if(ptr_outarg -> outrank != from_outptr -> out_rank)
        {
            fprintf(stderr,"\nError: Different mass of output arguments at
            call: [%s].\n\n",from_proc);
            exit(-1);
        }

        ptr_outarg = stack_out_args;
        from_outptr = from_outptr_help;
    }
    else
    {
        fprintf(stderr,"\nError: Nac Virtual Machine does not support
        Recursion.\n");
        exit(-1);
    }

    while(ptr_outarg != NULL)
    {
        to_lvptr = to_lvptr_help;
        while (to_lvptr != NULL)
        {
           if(strcmp(to_lvptr -> varname,ptr_outarg -> arg) == 0)
           {
               if(to_lvptr -> array_size == from_outptr -> array_size)
               {
                   found = 1;
                   tmp_from = from_outptr -> array;
                   tmp_to = to_lvptr -> array;
                   while(1)
                   {
                       tmp_to -> index = tmp_from -> index;
                       tmp_to -> value = tmp_from -> value;

                       tmp_from = tmp_from -> next;
                       tmp_to =  tmp_to -> next;
                       counter++;
                       if (counter == to_lvptr -> array_size)
                       {
                           counter = 0;
                           break;
                       }
                    }
                   from_outptr = from_outptr -> next;
              }
```

```
         else
         {
            fprintf(stderr,"\nError: Variables [%s] and [%s] have
            different array size at call: [%s].\n",to_lvptr->varname,
            from_outptr->varname,from_proc);
            exit(-1);
          }
      }
      to_lvptr = to_lvptr -> next;
}

//Αυξάνουμε τη ταχύτητα του NacVM.
//Δε χρειάζεται να ελέγξει τα παρακάτω.
if(found == 1)
{
    ptr_outarg = ptr_outarg -> next;
    found = 0;
    continue;
}

to_outptr = to_outptr_help;
while (to_outptr != NULL)
{
   if(strcmp(to_outptr -> varname,ptr_outarg -> arg) == 0)
   {
       if(to_outptr -> array_size == from_outptr -> array_size)
       {
           found = 1;
           tmp_from = from_outptr -> array;
           tmp_to = to_outptr -> array;
           while(1)
           {
               tmp_to -> index = tmp_from -> index;
               tmp_to -> value = tmp_from -> value;

               tmp_from = tmp_from -> next;
               tmp_to = tmp_to -> next;
               counter++;
               if (counter == from_outptr -> array_size)
               {
                  counter = 0;
                  break;
               }
           }
           from_outptr = from_outptr -> next;
       }
       else
       {
           fprintf(stderr,"\nError: Variables [%s] and [%s] have
           different array size at call: [%s].\n",to_outptr->varname,
           from_outptr->varname,from_proc);
           exit(-1);
       }
   }
   to_outptr = to_outptr -> next;
}

//Αυξάνουμε τη ταχύτητα του NacVM
//Δε χρειάζεται να ελέγξει τα παρακάτω
```

```
    if(found == 1)
    {
        ptr_outarg = ptr_outarg -> next;
        found = 0;
        continue;
    }

    glptr = ghead;
    while (glptr != NULL)
    {
       if(strcmp(glptr->varname,ptr_outarg->arg) == 0)
       {
           found = 1;
           if (glptr->array_size == from_outptr->array_size)
           {
               tmp_from = from_outptr -> array;
               tmp_to = glptr -> array;
               while(1)
               {
                   tmp_to -> index = tmp_from -> index;
                   tmp_to -> value = tmp_from -> value;

                   tmp_from = tmp_from -> next;
                   tmp_to = tmp_to -> next;
                   counter++;
                   if (counter == glptr->array_size)
                   {
                      counter = 0;
                      break;
                   }
               }
               from_outptr = from_outptr -> next;
           }
           else
           {
               fprintf(stderr,"\nError: Variables [%s] and [%s] have
               different array size at call: [%s].\n",glptr->varname,
               from_outptr->varname,from_proc);
               exit(-1);
           }
       }
       glptr = glptr -> next;
    }

    if(found == 0)
    {
        fprintf(stderr,"\nError: Variable [%s] at procedure [%s] has not
        defined.\n",ptr_outarg->arg,to_proc);
        exit(-1);
    }
    else
        found = 0;

    ptr_outarg = ptr_outarg -> next;
 }
 return 1;
}
```

```
// check_output_value
int check_output_value(int *proc_out_value, int *file_out_value, unsigned
long line)
{
    short static enter = 0;
    unsigned long static temp_line = 0;

    if(enter == 0)
    {
        fprintf(stdout,"\n**********NacVM RESULTS**********\n
        _____\n\nNac File: [%s]\n
        Input Data File: [%s]\n\n",NAC_FILE_NAME,INPUT_DATA_FNAME);
        temp_line = line;
    }

    if((*proc_out_value == *file_out_value) && (enter == 0))
    {
        fprintf(stdout,"\n%ld: Correct",line);
        enter = 1;
        return 0;
    }

    if((*proc_out_value != *file_out_value) && (enter == 0))
    {
        fprintf(stdout,"\n%ld: Wrong[%x]",line,*proc_out_value);
        enter = 1;
        return 0;
    }

    if(temp_line != line)
    {
        if(*proc_out_value == *file_out_value)
        {
            fprintf(stdout,"\n%ld: Correct",line);
            temp_line = line;
        }
        if(*proc_out_value != *file_out_value)
        {
            fprintf(stdout,"\n%ld: Wrong[%x]",line,*proc_out_value);
            temp_line = line;
        }
    }
    else
    {
        if (*proc_out_value == *file_out_value)
            fprintf(stdout," Correct");
        else
            fprintf(stdout," Wrong[%x]",*proc_out_value);
    }

    return 0;
}

// is_number
int is_number (char *str)
{
    short ctr = 0, is = 1, neg = 0;
    while (str[ctr] != '\0')
    {
```

```
        if (ctr == 0)
        {
            if (str[ctr] == '-')
            {
                ctr++;
                neg = 1;

                if(str[ctr] == '\0' )
                    break;
            }
        }
        if(isdigit(str[ctr]) == 0)
        {
            is = 0;
            break;
        }
        ctr++;
    }
    if (is == 1 && neg == 0)
        return strtol(str, (char **)NULL, 10);
    else if (is == 1 && neg == 1)
    {
          return (int)strtol(str, (char **)NULL, 10);
    }
    else
        return INT_MIN;
}

// fetch_execute_cycle
//Η "καρδιά" του NacVM
int fetch_execute_cycle()
{
    instruction *pc;
    range_loc *rl;
    rl = range_head;

    inarg *ptr_inarg;
    outarg *ptr_outarg;

    file *fh;
    fh = file_head;

    proc_list_acc *pla;
    pla = proc_head;

    unsigned long times = 1;
    int temp = 0, temp2 = 0;

    extern unsigned long _LINES_;

    while(fh -> next != NULL)
    {
        if ((pass_by_value_in(fh->in)) != 0)
        {
            fprintf(stderr,"\n Unable to pass the input values from file:
            [%s].\n", INPUT_DATA_FNAME);
            exit(-1);
        }
```

```
pc = head;
while(1)
{
    if (pc == NULL)
    {
        if (clist_tail -> to_procedure != NULL)
        {
            pc = clist_tail -> from_address;
            copy_proc_out_args(clist_tail -> to_procedure, pc ->
            procname, pc -> out);
            pc = pc -> next;
            if (clist_tail == clist_head)
                clist_tail -> to_procedure = NULL;
            else
                clist_tail = clist_tail -> previous;
        }
        //Αν η διαδικασία που κλήθηκε είναι η τελευταία του
        //αρχείου του κώδικα.
        else
            break;
    }
    else
    {
        if((strcmp(pc -> procname, head -> procname)!=0) &&
        clist_tail->to_procedure == NULL)
        {
            pc = tail;
            break;
        }
        if (clist_tail -> to_procedure != NULL)
        {
            if(strcmp(pc -> procname, clist_tail->to_procedure)!= 0)
            {
                pc = clist_tail -> from_address;
                copy_proc_out_args(clist_tail -> to_procedure, pc ->
                                                procname, pc -> out);
                pc = pc -> next;

                if (clist_tail == clist_head)
                    clist_tail -> to_procedure = NULL;
                else
                    clist_tail = clist_tail -> previous;
            }
        }
    }

    if(pc -> pcall == '0') //Κλήση διαδικασίας
    {
        ptr_inarg = pc -> in;
        ptr_outarg = pc -> out;

        if (strcmp(pc -> op, "nop") == 0)
        {
            pc = pc -> next;
        }
        else if (strcmp(pc -> op,"mov") == 0)
        {
            if ((temp = is_number(ptr_inarg -> arg)) > INT_MIN)
            {
```

```
            if((_mov_(value(pc -> procname,ptr_outarg ->
                                    arg,0),&temp)) == 1);
            else
            {
                fprintf(stderr,"\nError: Failure at statement
                            [mov] at line: %ld.\n",_LINES_);
                exit(-1);
            }
        }
        else
        {
            if((_mov_(value(pc -> procname,ptr_outarg -> arg,0)
            ,value(pc -> procname,ptr_inarg -> arg,0))) == 1);
            else
            {
                fprintf(stderr,"\nError: Failure at statement
                [mov] at line: %ld.\n",_LINES_);
                exit(-1);
            }
        }
        pc = pc -> next;

    }
    else if (strcmp(pc -> op,"ldc") == 0)
    {
        if ((temp = is_number(ptr_inarg -> arg)) > INT_MIN)
        {
            if((_ldc_(value(pc->procname,ptr_outarg->arg,0),
                                              &temp)) == 1);
            else
            {
                fprintf(stderr,"\nError: Failure at statement
                              [ldc] at line: %ld.\n",_LINES_);
                exit(-1);
            }
        }
        else
        {
             if((_ldc_(value(pc -> procname,ptr_outarg -> arg,0)
              ,value(pc -> procname,ptr_inarg -> arg,0))) == 1);
            else
            {
                fprintf(stderr,"\nError: Failure at statement
                                [ldc] at line: %ld.\n",_LINES_);
                exit(-1);
            }
        }
        pc = pc -> next;

    }
    else if (strcmp (pc -> op, "jmpun") == 0)
    {
        pc = _jmpun_(ptr_outarg -> arg, pc -> procname);

    }
    else if (strcmp(pc -> op, "jmpeq") == 0)
    {
        if((temp=is_number(ptr_inarg -> next -> arg)) > INT_MIN)
            pc = _jmpeq_(ptr_outarg->arg,ptr_outarg->next-> arg,
```

```
            pc -> procname, value(pc->procname,ptr_inarg -> arg,0), &temp);
                  else
                       pc = _jmpeq_(ptr_outarg->arg,ptr_outarg->next-> arg,
  pc -> procname, value(pc->procname,ptr_inarg -> arg,0),value(pc->procname,
                                              ptr_inarg -> next-> arg,0));

               }
               else if (strcmp(pc -> op, "jmpne") == 0)
               {
                   if((temp=is_number(ptr_inarg -> next -> arg)) > INT_MIN)
                       pc = _jmpne_(ptr_outarg->arg,ptr_outarg->next->arg,
                       pc -> procname, value(pc->procname,ptr_inarg->arg
                                                          ,0),&temp);
                   else
                       pc=_jmpne_(ptr_outarg->arg,ptr_outarg->next->arg, pc
                                -> procname, value(pc->procname,ptr_inarg ->
                         arg,0),value(pc->procname,ptr_inarg -> next-> arg,0));

               }
               else if (strcmp(pc -> op, "jmplt") == 0)
               {
                   if((temp=is_number(ptr_inarg -> next -> arg)) > INT_MIN)
                       pc=_jmplt_(ptr_outarg->arg,ptr_outarg->next->arg, pc
               -> procname, value(pc->procname,ptr_inarg -> arg,0), &temp);
                   else
                       pc = _jmplt_(ptr_outarg -> arg, ptr_outarg -> next -
> arg, pc -> procname, value(pc->procname,ptr_inarg -> arg,0),value(pc-
>procname,ptr_inarg -> next-> arg,0));

               }
               else if (strcmp(pc -> op, "jmple") == 0)
               {
                   if((temp=is_number(ptr_inarg -> next -> arg)) > INT_MIN)
                       pc = _jmple_(ptr_outarg -> arg, ptr_outarg -> next -
> arg, pc -> procname, value(pc->procname,ptr_inarg -> arg,0), &temp);
                   else
                       pc = _jmple_(ptr_outarg -> arg, ptr_outarg -> next -
> arg, pc -> procname, value(pc->procname,ptr_inarg -> arg,0),value(pc-
>procname,ptr_inarg -> next-> arg,0));

               }
               else if (strcmp(pc -> op, "jmpgt") == 0)
               {
                   if((temp=is_number(ptr_inarg -> next -> arg)) > INT_MIN)
                       pc = _jmpgt_(ptr_outarg -> arg, ptr_outarg -> next -
> arg, pc -> procname, value(pc->procname,ptr_inarg -> arg,0), &temp);
                   else
                       pc = _jmpgt_(ptr_outarg -> arg, ptr_outarg -> next -
> arg, pc -> procname, value(pc->procname,ptr_inarg -> arg,0),value(pc-
>procname,ptr_inarg -> next-> arg,0));

               }
               else if (strcmp(pc -> op, "jmpge") == 0)
               {
                   if((temp=is_number(ptr_inarg -> next -> arg)) > INT_MIN)
                       pc = _jmpge_(ptr_outarg -> arg, ptr_outarg -> next -
> arg, pc -> procname, value(pc->procname,ptr_inarg -> arg,0), &temp);
                   else
```

```
                        pc = _jmpge_(ptr_outarg -> arg, ptr_outarg -> next -
> arg, pc -> procname, value(pc->procname,ptr_inarg -> arg,0),value(pc-
>procname,ptr_inarg -> next-> arg,0));

                }
                else if (strcmp(pc -> op,"add") == 0)
                {
                    if ((temp = is_number(ptr_inarg -> arg)) > INT_MIN)
                    {
                        if((temp2=is_number(ptr_inarg->next->arg))> INT_MIN)
                        {
                            if((_add_(value(pc->procname,ptr_outarg->arg,0),
                                                    &temp,&temp2)) == 1);
                            else
                            {
                                fprintf(stderr,"\nError: Failure at
                               statement [add] at line: %ld.\n",_LINES_);
                                exit(-1);
                            }
                        }
                        else
                        {
                            if((_add_(value(pc -> procname,ptr_outarg ->
arg,0),&temp,value(pc -> procname,ptr_inarg -> next -> arg,0))) == 1);
                            else
                            {
                                fprintf(stderr,"\nError: Failure at
statement [add] at line: %ld.\n",_LINES_);
                                exit(-1);
                            }
                        }
                    }
                    else
                    {
                        if((temp2=is_number(ptr_inarg->next->arg))>INT_MIN)
                        {
                            if((_add_(value(pc -> procname,ptr_outarg ->
arg,0),value(pc -> procname,ptr_inarg -> arg,0),&temp2)) == 1);
                            else
                            {
                                fprintf(stderr,"\nError: Failure at
statement [add] at line: %ld.\n",_LINES_);
                                exit(-1);
                            }
                        }
                        else
                        {
                            if((_add_(value(pc -> procname,ptr_outarg ->
arg,0),value(pc -> procname,ptr_inarg -> arg,0),value(pc ->
procname,ptr_inarg -> next-> arg,0))) == 1);
                            else
                            {
                                fprintf(stderr,"\nError: Failure at
statement [add] at line: %ld.\n",_LINES_);
                                exit(-1);
                            }
                        }
                    }
                    pc = pc -> next;
```

```
}
else if (strcmp(pc -> op,"sub") == 0)
{
    if ((temp = is_number(ptr_inarg -> arg)) > INT_MIN)
    {
        if((temp2=is_number(ptr_inarg->next->arg))>INT_MIN)
        {
            if((_sub_(value(pc->procname,ptr_outarg->arg,0),
                                    &temp,&temp2)) == 1);
            else
            {
                fprintf(stderr,"\nError: Failure at
               statement [sub] at line: %ld.\n",_LINES_);
                exit(-1);
            }
        }
        else
        {
            if((_sub_(value(pc->procname,ptr_outarg->arg,
             0),&temp,value(pc->procname,ptr_inarg->next->
                                          arg,0))) == 1);
            else
            {
                fprintf(stderr,"\nError: Failure at
                 statement [sub] at line: %ld.\n",_LINES_);
                exit(-1);
            }
        }
    }
    else
    {
        if((temp2=is_number(ptr_inarg->next->arg))>INT_MIN)
        {
            if((_sub_(value(pc -> procname,ptr_outarg ->
             arg,0),value(pc -> procname,ptr_inarg ->
                                   arg,0),&temp2)) == 1);
            else
            {
                fprintf(stderr,"\nError: Failure at
                statement [sub] at line: %ld.\n",_LINES_);
                exit(-1);
            }
        }
        else
        {
            if((_sub_(value(pc->procname,ptr_outarg->
             arg,0),value(pc->procname,ptr_inarg->arg,
             0),value(pc -> procname,ptr_inarg -> next->
                                        arg,0))) == 1);
            else
            {
                fprintf(stderr,"\nError: Failure at
                statement [sub] at line: %ld.\n",_LINES_);
                exit(-1);
            }
        }
    }
    pc = pc -> next;
}
```

```
                else if (strcmp(pc -> op,"seteq") == 0)
                {
                    if ((temp = is_number(ptr_inarg -> arg)) > INT_MIN)
                    {
                        if((temp2=is_number(ptr_inarg->next->arg))>INT_MIN)
                        {
                            if((_seteq_(value(pc -> procname,ptr_outarg ->
                                            arg,0),&temp,&temp2)) == 1);
                            else
                            {
                                fprintf(stderr,"\nError: Failure at
                                statement [seteq] at line: %ld.\n",_LINES_);
                                exit(-1);
                            }
                        }
                        else
                        {
                            if((_seteq_(value(pc->procname,ptr_outarg->arg,
          0),&temp,value(pc -> procname,ptr_inarg -> next -> arg,0))) == 1);
                            else
                            {
                                fprintf(stderr,"\nError: Failure at
                                statement [seteq] at line: %ld.\n",_LINES_);
                                exit(-1);
                            }
                        }
                    }
                    else
                    {
                        if((temp2=is_number(ptr_inarg->next->arg))> INT_MIN)
                        {
                            if((_seteq_(value(pc -> procname,ptr_outarg ->
arg,0),value(pc -> procname,ptr_inarg -> arg,0),&temp2)) == 1);
                            else
                            {
                                fprintf(stderr,"\nError: Failure at
statement [seteq] at line: %ld.\n",_LINES_);
                                exit(-1);
                            }
                        }
                        else
                        {
                            if((_seteq_(value(pc -> procname,ptr_outarg ->
arg,0),value(pc -> procname,ptr_inarg -> arg,0),value(pc ->
procname,ptr_inarg -> next-> arg,0))) == 1);
                            else
                            {
                                fprintf(stderr,"\nError: Failure at
statement [seteq] at line: %ld.\n",_LINES_);
                                exit(-1);
                            }
                        }
                    }
                    pc = pc -> next;
                }
                else if (strcmp(pc -> op,"setne") == 0)
                {
                    if ((temp = is_number(ptr_inarg -> arg)) > INT_MIN)
                    {
```

```
                        if((temp2=is_number(ptr_inarg->next->arg))>INT_MIN)
                        {
                            if((_setne_(value(pc -> procname,ptr_outarg ->
arg,0),&temp,&temp2)) == 1);
                            else
                            {
                                fprintf(stderr,"\nError: Failure at
statement [setne] at line: %ld.\n",_LINES_);
                                exit(-1);
                            }
                        }
                        else
                        {
                            if((_setne_(value(pc -> procname,ptr_outarg ->
arg,0),&temp,value(pc -> procname,ptr_inarg -> next -> arg,0))) == 1);
                            else
                            {
                                fprintf(stderr,"\nError: Failure at
statement [setne] at line: %ld.\n",_LINES_);
                                exit(-1);
                            }
                        }
                    }
                    else
                    {
                        if((temp2=is_number(ptr_inarg->next->arg))>INT_MIN)
                        {
                            if((_setne_(value(pc -> procname,ptr_outarg ->
arg,0),value(pc -> procname,ptr_inarg -> arg,0),&temp2)) == 1);
                            else
                            {
                                fprintf(stderr,"\nError: Failure at
statement [setne] at line: %ld.\n",_LINES_);
                                exit(-1);
                            }
                        }
                        else
                        {
                            if((_setne_(value(pc -> procname,ptr_outarg ->
arg,0),value(pc -> procname,ptr_inarg -> arg,0),value(pc ->
procname,ptr_inarg -> next-> arg,0))) == 1);
                            else
                            {
                                fprintf(stderr,"\nError: Failure at
statement [setne] at line: %ld.\n",_LINES_);
                                exit(-1);
                            }
                        }
                    }
                    pc = pc -> next;
                }
                else if (strcmp(pc -> op,"setlt") == 0)
                {
                    if ((temp = is_number(ptr_inarg -> arg)) > INT_MIN)
                    {
                        if((temp2=is_number(ptr_inarg->next->arg))>INT_MIN)
                        {
                            if((_setlt_(value(pc -> procname,ptr_outarg ->
arg,0),&temp,&temp2)) == 1);
```

```
                            else
                            {
                                fprintf(stderr,"\nError: Failure at
statement [setlt] at line: %ld.\n",_LINES_);
                                exit(-1);
                            }
                        }
                        else
                        {
                            if((_setlt_(value(pc -> procname,ptr_outarg ->
arg,0),&temp,value(pc -> procname,ptr_inarg -> next -> arg,0))) == 1);
                            else
                            {
                                fprintf(stderr,"\nError: Failure at
statement [setlt] at line: %ld.\n",_LINES_);
                                exit(-1);
                            }
                        }
                    }
                    else
                    {
                        if((temp2=is_number(ptr_inarg->next->arg))>INT_MIN)
                        {
                            if((_setlt_(value(pc -> procname,ptr_outarg ->
arg,0),value(pc -> procname,ptr_inarg -> arg,0),&temp2)) == 1);
                            else
                            {
                                fprintf(stderr,"\nError: Failure at
statement [setlt] at line: %ld.\n",_LINES_);
                                exit(-1);
                            }
                        }
                        else
                        {
                            if((_setlt_(value(pc -> procname,ptr_outarg ->
arg,0),value(pc -> procname,ptr_inarg -> arg,0),value(pc ->
procname,ptr_inarg -> next-> arg,0))) == 1);
                            else
                            {
                                fprintf(stderr,"\nError: Failure at
statement [setlt] at line: %ld.\n",_LINES_);
                                exit(-1);
                            }
                        }
                    }
                    pc = pc -> next;
                }
                else if (strcmp(pc -> op,"setle") == 0)
                {
                    if ((temp = is_number(ptr_inarg -> arg)) > INT_MIN)
                    {
                        if((temp2=is_number(ptr_inarg->next->arg))>INT_MIN)
                        {
                            if((_setle_(value(pc -> procname,ptr_outarg ->
arg,0),&temp,&temp2)) == 1);
                            else
                            {
                                fprintf(stderr,"\nError: Failure at
statement [setle] at line: %ld.\n",_LINES_);
```

```
                                exit(-1);
                            }
                        }
                        else
                        {
                            if((_setle_(value(pc -> procname,ptr_outarg ->
arg,0),&temp,value(pc -> procname,ptr_inarg -> next -> arg,0))) == 1);
                            else
                            {
                                fprintf(stderr,"\nError: Failure at
statement [setle] at line: %ld.\n",_LINES_);
                                exit(-1);
                            }
                        }
                    }
                    else
                    {
                        if((temp2=is_number(ptr_inarg->next->arg))>INT_MIN)
                        {
                            if((_setle_(value(pc -> procname,ptr_outarg ->
arg,0),value(pc -> procname,ptr_inarg -> arg,0),&temp2)) == 1);
                            else
                            {
                                fprintf(stderr,"\nError: Failure at
statement [setle] at line: %ld.\n",_LINES_);
                                exit(-1);
                            }
                        }
                        else
                        {
                            if((_setle_(value(pc -> procname,ptr_outarg ->
arg,0),value(pc -> procname,ptr_inarg -> arg,0),value(pc ->
procname,ptr_inarg -> next-> arg,0))) == 1);
                            else
                            {
                                fprintf(stderr,"\nError: Failure at
statement [setle] at line: %ld.\n",_LINES_);
                                exit(-1);
                            }
                        }
                    }
                    pc = pc -> next;
                }
                else if (strcmp(pc -> op,"setgt") == 0)
                {
                    if ((temp = is_number(ptr_inarg -> arg)) > INT_MIN)

                        if((temp2=is_number(ptr_inarg->next->arg))>INT_MIN)
                        {
                            if((_setgt_(value(pc->procname,ptr_outarg->arg,
                                                0),&temp,&temp2)) == 1);
                            else
                            {
                                fprintf(stderr,"\nError: Failure at
                                statement [setgt] at line: %ld.\n",_LINES_);
                                exit(-1);
                            }
                        }
                        else
```

```
                        {
                            if((_setgt_(value(pc -> procname,ptr_outarg ->
arg,0),&temp,value(pc -> procname,ptr_inarg -> next -> arg,0))) == 1);
                            else
                            {
                                fprintf(stderr,"\nError: Failure at
statement [setgt] at line: %ld.\n",_LINES_);
                                exit(-1);
                            }
                        }
                    }
                    else
                    {
                        if((temp2=is_number(ptr_inarg->next->arg))>INT_MIN)
                        {
                            if((_setgt_(value(pc -> procname,ptr_outarg ->
arg,0),value(pc -> procname,ptr_inarg -> arg,0),&temp2)) == 1);
                            else
                            {
                                fprintf(stderr,"\nError: Failure at
statement [setgt] at line: %ld.\n",_LINES_);
                                exit(-1);
                            }
                        }
                        else
                        {
                            if((_setgt_(value(pc -> procname,ptr_outarg ->
arg,0),value(pc -> procname,ptr_inarg -> arg,0),value(pc ->
procname,ptr_inarg -> next-> arg,0))) == 1);
                            else
                            {
                                fprintf(stderr,"\nError: Failure at
statement [setgt] at line: %ld.\n",_LINES_);
                                exit(-1);
                            }
                        }
                    }
                    pc = pc -> next;

                }
                else if (strcmp(pc -> op,"setge") == 0)
                {
                    if ((temp = is_number(ptr_inarg -> arg)) > INT_MIN)
                    {
                        if((temp2=is_number(ptr_inarg->next->arg))>INT_MIN)
                        {
                            if((_setge_(value(pc -> procname,ptr_outarg ->
arg,0),&temp,&temp2)) == 1);
                            else
                            {
                                fprintf(stderr,"\nError: Failure at
statement [setge] at line: %ld.\n",_LINES_);
                                exit(-1);
                            }
                        }
                        else
                        {
                            if((_setge_(value(pc -> procname,ptr_outarg ->
arg,0),&temp,value(pc -> procname,ptr_inarg -> next -> arg,0))) == 1);
```

```
                            else
                            {
                                fprintf(stderr,"\nError: Failure at
statement [setge] at line: %ld.\n",_LINES_);
                                exit(-1);
                            }
                        }
                    }
                    else
                    {
                        if((temp2=is_number(ptr_inarg->next->arg))>INT_MIN)
                        {
                            if((_setge_(value(pc -> procname,ptr_outarg ->
arg,0),value(pc -> procname,ptr_inarg -> arg,0),&temp2)) == 1);
                            else
                            {
                                fprintf(stderr,"\nError: Failure at
statement [setge] at line: %ld.\n",_LINES_);
                                exit(-1);
                            }
                        }
                        else
                        {
                            if((_setge_(value(pc -> procname,ptr_outarg ->
arg,0),value(pc -> procname,ptr_inarg -> arg,0),value(pc ->
procname,ptr_inarg -> next-> arg,0))) == 1);
                            else
                            {
                                fprintf(stderr,"\nError: Failure at
statement [setge] at line: %ld.\n",_LINES_);
                                exit(-1);
                            }
                        }
                    }
                    pc = pc -> next;
                }
                else if(strcmp(pc -> op,"load") == 0)
                {
                     if ((temp=is_number(ptr_inarg->next->arg))>INT_MIN)
                     {
                        if((_load_(value(pc -> procname,ptr_outarg ->
arg,0),pc -> procname, ptr_inarg -> arg,&temp)) == 1);
                        else
                        {
                            fprintf(stderr,"\nError: Failure at statement
[load] at line: %ld.\n", _LINES_);
                            exit(-1);
                        }
                     }
                     else
                     {
                         if((_load_(value(pc -> procname,ptr_outarg ->
arg,0),pc -> procname, ptr_inarg -> arg,value(pc -> procname,ptr_inarg ->
next -> arg,0))) == 1);
                        else
                        {
                            fprintf(stderr,"\nError: Failure at statement
[load] at line: %ld.\n", _LINES_);
                            exit(-1);
```

```
                        }
                    }
                    pc = pc -> next;
                }
                else if(strcmp(pc -> op,"store") == 0)
                {
                    if((temp=is_number(ptr_inarg->next->arg))>INT_MIN)
                    {
                       if((_store_(pc -> procname,ptr_outarg ->
arg,value(pc -> procname,ptr_inarg -> arg,0),&temp)) == 1);
                       else
                       {
                           fprintf(stderr,"\nError: Failure at statement
[store] at line: %ld.\n", _LINES_);
                           exit(-1);
                       }
                    }
                    else
                    {
                        if((_store_(pc -> procname, ptr_outarg -> arg,
value(pc -> procname,ptr_inarg -> arg,0),value(pc -> procname,ptr_inarg ->
next -> arg,0))) == 1);
                       else
                       {
                           fprintf(stderr,"\nError: Failure at statement
[store] at line: %ld.\n", _LINES_);
                           exit(-1);
                       }
                    }
                    pc = pc -> next;
                }
                else if(strcmp(pc -> op,"print") == 0)
                {
                    if ((temp = is_number(ptr_inarg -> arg)) > INT_MIN)
                    {
                       fprintf(stderr,"\nError: Failure at statement
[print] at line: %ld. The argument [%s] must be a variable.\n",
_LINES_,ptr_inarg -> arg);
                       exit(-1);
                    }
                    else
                    {
                       if((_print_(pc -> procname,ptr_inarg -> arg)) == 1);
                       else
                       {
                           fprintf(stderr,"\nError: Failure at statement
[print] at line: %ld.\n", _LINES_);
                           exit(-1);
                       }
                    }
                    pc = pc -> next;
                }
                else if(strcmp(pc -> op,"nand") == 0)
                {
                    if((temp=is_number(ptr_inarg->next->arg))>INT_MIN)
                    {
                        if(_nand_(value(pc -> procname,ptr_outarg ->
arg,0),value(pc -> procname,ptr_inarg -> arg,0),&temp) == 1);
                       else
```

```
                        {
                            fprintf(stderr,"\nError: Failure at statement
[nand] at line: %ld.\n", _LINES_);
                            exit(-1);
                        }
                    }
                    else
                    {
                       if(_nand_(value(pc -> procname,ptr_outarg ->
arg,0),value(pc -> procname,ptr_inarg -> arg,0),value(pc ->
procname,ptr_inarg -> next -> arg, 0)) == 1);
                        else
                        {
                            fprintf(stderr,"\nError: Failure at statement
[nand] at line: %ld.\n", _LINES_);
                            exit(-1);
                        }
                    }
                    pc = pc -> next;
                }
                else if(strcmp(pc -> op,"ior") == 0)
                {
                    if((temp=is_number(ptr_inarg->next->arg))>INT_MIN)
                    {
                        if(_ior_(value(pc -> procname,ptr_outarg ->
arg,0),value(pc -> procname,ptr_inarg -> arg,0),&temp) == 1);
                        else
                        {
                            fprintf(stderr,"\nError: Failure at statement
[ior] at line: %ld.\n", _LINES_);
                            exit(-1);
                        }
                    }
                    else
                    {
                       if(_ior_(value(pc -> procname,ptr_outarg ->
arg,0),value(pc -> procname,ptr_inarg -> arg,0),value(pc ->
procname,ptr_inarg -> next -> arg, 0)) == 1);
                        else
                        {
                            fprintf(stderr,"\nError: Failure at statement
[ior] at line: %ld.\n", _LINES_);
                            exit(-1);
                        }
                    }
                    pc = pc -> next;
                }
                else if(strcmp(pc -> op,"xor") == 0)
                {
                    if((temp=is_number(ptr_inarg->next->arg))>INT_MIN)
                    {
                        if(_xor_(value(pc -> procname,ptr_outarg ->
arg,0),value(pc -> procname,ptr_inarg -> arg,0),&temp) == 1);
                        else
                        {
                            fprintf(stderr,"\nError: Failure at statement
[xor] at line: %ld.\n", _LINES_);
                            exit(-1);
                        }
```

```
                    }
                    else
                    {
                       if(_xor_(value(pc -> procname,ptr_outarg ->
arg,0),value(pc -> procname,ptr_inarg -> arg,0),value(pc ->
procname,ptr_inarg -> next -> arg, 0)) == 1);
                       else
                       {
                           fprintf(stderr,"\nError: Failure at statement
[xor] at line: %ld.\n", _LINES_);
                           exit(-1);
                       }
                    }
                    pc = pc -> next;
               }
               else if(strcmp(pc -> op,"and") == 0)
               {
                    if((temp=is_number(ptr_inarg->next->arg))>INT_MIN)
                    {
                        if(_and_(value(pc -> procname,ptr_outarg ->
arg,0),value(pc -> procname,ptr_inarg -> arg,0),&temp) == 1);
                       else
                       {
                           fprintf(stderr,"\nError: Failure at statement
[and] at line: %ld.\n", _LINES_);
                           exit(-1);
                       }
                    }
                    else
                    {
                       if(_and_(value(pc -> procname,ptr_outarg ->
arg,0),value(pc -> procname,ptr_inarg -> arg,0),value(pc ->
procname,ptr_inarg -> next -> arg, 0)) == 1);
                       else
                       {
                           fprintf(stderr,"\nError: Failure at statement
[and] at line: %ld.\n", _LINES_);
                           exit(-1);
                       }
                    }
                    pc = pc -> next;
               }
               else if(strcmp(pc -> op,"nor") == 0)
               {
                    if((temp=is_number(ptr_inarg->next->arg))>INT_MIN)
                    {
                       if(_nor_(value(pc -> procname,ptr_outarg ->
arg,0),value(pc -> procname,ptr_inarg -> arg,0),&temp) == 1);
                       else
                       {
                           fprintf(stderr,"\nError: Failure at statement
[nor] at line: %ld.\n", _LINES_);
                           exit(-1);
                       }
                    }
                    else
                    {
```

```
                        if(_nor_(value(pc -> procname,ptr_outarg ->
arg,0),value(pc -> procname,ptr_inarg -> arg,0),value(pc ->
procname,ptr_inarg -> next -> arg, 0)) == 1);
                        else
                        {
                            fprintf(stderr,"\nError: Failure at statement
[nor] at line: %ld.\n", _LINES_);
                            exit(-1);
                        }
                    }
                    pc = pc -> next;
                }
                else if(strcmp(pc -> op,"xnor") == 0)
                {
                    if((temp=is_number(ptr_inarg->next->arg))>INT_MIN)
                    {
                        if(_xnor_(value(pc -> procname,ptr_outarg ->
arg,0),value(pc -> procname,ptr_inarg -> arg,0),&temp) == 1);
                        else
                        {
                            fprintf(stderr,"\nError: Failure at statement
[xnor] at line: %ld.\n", _LINES_);
                            exit(-1);
                        }
                    }
                    else
                    {
                        if(_xnor_(value(pc -> procname,ptr_outarg ->
arg,0),value(pc -> procname,ptr_inarg -> arg,0),value(pc ->
procname,ptr_inarg -> next -> arg, 0)) == 1);
                        else
                        {
                            fprintf(stderr,"\nError: Failure at statement
[xnor] at line: %ld.\n", _LINES_);
                            exit(-1);
                        }
                    }
                    pc = pc -> next;
                }
                else if(strcmp(pc -> op,"not") == 0)
                {
                    if ((temp = is_number(ptr_inarg -> arg)) > INT_MIN)
                    {
                        if(_not_(value(pc->procname,ptr_outarg->arg,0),
&temp) == 1);
                        else
                        {
                            fprintf(stderr,"\nError: Failure at statement
[not] at line: %ld.\n", _LINES_);
                            exit(-1);
                        }
                    }
                    else
                    {
                        if(_not_(value(pc -> procname,ptr_outarg ->
arg,0),value(pc -> procname,ptr_inarg -> arg,0)) == 1);
                        else
                        {
```

```
                            fprintf(stderr,"\nError: Failure at statement
[not] at line: %ld.\n", _LINES_);
                            exit(-1);
                        }
                    }
                    pc = pc -> next;
                }
                else if(strcmp(pc -> op,"div") == 0)
                {
                     if((temp=is_number(ptr_inarg->next -> arg)) > INT_MIN)
                     {
                         if(_div_(value(pc -> procname,ptr_outarg ->
arg,0),value(pc -> procname,ptr_inarg -> arg,0),&temp) == 1);
                        else
                        {
                            fprintf(stderr,"\nError: Failure at statement
[div] at line: %ld.\n", _LINES_);
                            exit(-1);
                        }
                     }
                     else
                     {
                        if(_div_(value(pc -> procname,ptr_outarg ->
arg,0),value(pc -> procname,ptr_inarg -> arg,0),value(pc ->
procname,ptr_inarg -> next -> arg, 0)) == 1);
                        else
                        {
                            fprintf(stderr,"\nError: Failure at statement
[div] at line: %ld.\n", _LINES_);
                            exit(-1);
                        }
                     }
                     pc = pc -> next;
                }
                else if(strcmp(pc -> op,"rem") == 0)
                {
                     if((temp=is_number(ptr_inarg->next->arg))>INT_MIN)
                     {
                         if(_rem_(value(pc -> procname,ptr_outarg ->
arg,0),value(pc -> procname,ptr_inarg -> arg,0),&temp) == 1);
                        else
                        {
                            fprintf(stderr,"\nError: Failure at statement
[rem] at line: %ld.\n", _LINES_);
                            exit(-1);
                        }
                     }
                     else
                     {
                        if(_rem_(value(pc -> procname,ptr_outarg ->
arg,0),value(pc -> procname,ptr_inarg -> arg,0),value(pc ->
procname,ptr_inarg -> next -> arg, 0)) == 1);
                        else
                        {
                            fprintf(stderr,"\nError: Failure at statement
[rem] at line: %ld.\n", _LINES_);
                            exit(-1);
                        }
                     }
```

```
                    pc = pc -> next;
                }
                else if(strcmp(pc -> op,"divrem") == 0)
                {
                    if((temp=is_number(ptr_inarg->next->arg))>INT_MIN)
                    {
                        if(_divrem_(value(pc -> procname,ptr_outarg ->
arg,0),value(pc -> procname,ptr_outarg -> next-> arg,0),value(pc ->
procname,ptr_inarg -> arg,0),&temp) == 1);
                        else
                        {
                            fprintf(stderr,"\nError: Failure at statement
[divrem] at line: %ld.\n", _LINES_);
                            exit(-1);
                        }
                    }
                    else
                    {
                        if(_divrem_(value(pc -> procname,ptr_outarg ->
arg,0),value(pc -> procname,ptr_outarg -> next -> arg,0),value(pc ->
procname,ptr_inarg -> arg,0),value(pc -> procname,ptr_inarg -> next -> arg,
0)) == 1);
                        else
                        {
                            fprintf(stderr,"\nError: Failure at statement
[divrem] at line: %ld.\n", _LINES_);
                            exit(-1);
                        }
                    }
                    pc = pc -> next;
                }
                else if(strcmp(pc -> op,"abs") == 0)
                {
                    if((_abs_(value(pc -> procname,ptr_outarg ->
arg,0),value(pc -> procname,ptr_inarg -> arg,0))) == 1);
                    else
                    {
                        fprintf(stderr,"\nError: Failure at statement [abs]
at line: %ld.\n", _LINES_);
                        exit(-1);
                    }
                    pc = pc -> next;
                }
                else if(strcmp(pc -> op,"neg") == 0)
                {
                    if((temp = is_number(ptr_inarg  -> arg)) > INT_MIN)
                    {
                        if(_neg_(value(pc -> procname,ptr_outarg ->
arg,0),&temp) == 1);
                        else
                        {
                            fprintf(stderr,"\nError: Failure at statement
[neg] at line: %ld.\n", _LINES_);
                            exit(-1);
                        }
                    }
                    else
                    {
```

```
                        if((_neg_(value(pc -> procname,ptr_outarg ->
arg,0),value(pc -> procname,ptr_inarg -> arg,0))) == 1);
                        else
                        {
                            fprintf(stderr,"\nError: Failure at statement
[neg] at line: %ld.\n", _LINES_);
                            exit(-1);
                        }
                    }
                    pc = pc -> next;
                }
                else if(strcmp(pc -> op,"zxt") == 0)
                {
                    if((_zxt_(value(pc -> procname,ptr_outarg ->
arg,0),value(pc -> procname,ptr_inarg -> arg,0))) == 1);
                    else
                    {
                        fprintf(stderr,"\nError: Failure at statement [zxt]
at line: %ld.\n", _LINES_);
                        exit(-1);
                    }

                    pc = pc -> next;
                }
                else if(strcmp(pc -> op,"sxt") == 0)
                {
                    if((_sxt_(value(pc -> procname,ptr_outarg ->
arg,0),value(pc -> procname,ptr_inarg -> arg,0))) == 1);
                    else
                    {
                        fprintf(stderr,"\nError: Failure at statement [sxt]
at line: %ld.\n", _LINES_);
                        exit(-1);
                    }

                    pc = pc -> next;

                }
                else if(strcmp(pc -> op,"trunc") == 0)
                {
                    if((_trunc_(value(pc -> procname,ptr_outarg ->
arg,0),value(pc -> procname,ptr_inarg -> arg,0))) == 1);
                    else
                    {
                        fprintf(stderr,"\nError: Failure at statement
[trunc] at line: %ld.\n", _LINES_);
                        exit(-1);
                    }

                    pc = pc -> next;
                }
                else if(strcmp(pc -> op,"min") == 0)
                {
                    if((temp=is_number(ptr_inarg -> next-> arg)) > INT_MIN)
                    {
                        if(_min_(value(pc -> procname,ptr_outarg ->
arg,0),value(pc -> procname,ptr_inarg -> arg,0),&temp) == 1);
                        else
                        {
```

```
                            fprintf(stderr,"\nError: Failure at statement
[min] at line: %ld.\n", _LINES_);
                            exit(-1);
                        }
                    }
                    else
                    {
                        if((_min_(value(pc -> procname,ptr_outarg ->
arg,0),value(pc -> procname,ptr_inarg -> arg,0),value(pc ->
procname,ptr_inarg -> next-> arg,0))) == 1);
                        else
                        {
                            fprintf(stderr,"\nError: Failure at statement
[min] at line: %ld.\n", _LINES_);
                            exit(-1);
                        }
                    }
                    pc = pc -> next;
                }
                else if(strcmp(pc -> op,"max") == 0)
                {
                    if((temp=is_number(ptr_inarg -> next -> arg)) > INT_MIN)
                    {
                        if(_max_(value(pc -> procname,ptr_outarg ->
arg,0),value(pc -> procname,ptr_inarg -> arg,0),&temp) == 1);
                        else
                        {
                            fprintf(stderr,"\nError: Failure at statement
[max] at line: %ld.\n", _LINES_);
                            exit(-1);
                        }
                    }
                    else
                    {
                        if((_max_(value(pc -> procname,ptr_outarg ->
arg,0),value(pc -> procname,ptr_inarg -> arg,0),value(pc ->
procname,ptr_inarg -> next -> arg,0))) == 1);
                        else
                        {
                            fprintf(stderr,"\nError: Failure at statement
[max] at line: %ld.\n", _LINES_);
                            exit(-1);
                        }
                    }
                    pc = pc -> next;
                }
                else if(strcmp(pc -> op,"shl") == 0)
                {
                    if((temp=is_number(ptr_inarg -> next -> arg)) > INT_MIN)
                    {
                        if(_shl_(value(pc -> procname,ptr_outarg ->
arg,0),value(pc -> procname,ptr_inarg -> arg,0),&temp) == 1);
                        else
                        {
                            fprintf(stderr,"\nError: Failure at statement
[shl] at line: %ld.\n", _LINES_);
                            exit(-1);
                        }
                    }
```

```
                    else
                    {
                        if((_shl_(value(pc -> procname,ptr_outarg ->
arg,0),value(pc -> procname,ptr_inarg -> arg,0),value(pc ->
procname,ptr_inarg -> next -> arg,0))) == 1);
                        else
                        {
                            fprintf(stderr,"\nError: Failure at statement
[shl] at line: %ld.\n", _LINES_);
                            exit(-1);
                        }
                    }
                    pc = pc -> next;
                }
                else if(strcmp(pc -> op,"shr") == 0)
                {
                    if((temp=is_number(ptr_inarg -> next -> arg)) > INT_MIN)
                    {
                        if(_shr_(value(pc -> procname,ptr_outarg ->
arg,0),value(pc -> procname,ptr_inarg -> arg,0),&temp) == 1);
                        else
                        {
                            fprintf(stderr,"\nError: Failure at statement
[shr] at line: %ld.\n", _LINES_);
                            exit(-1);
                        }
                    }
                    else
                    {
                        if((_shr_(value(pc -> procname,ptr_outarg ->
arg,0),value(pc -> procname,ptr_inarg -> arg,0),value(pc ->
procname,ptr_inarg -> next -> arg,0))) == 1);
                        else
                        {
                            fprintf(stderr,"\nError: Failure at statement
[shr] at line: %ld.\n", _LINES_);
                            exit(-1);
                        }
                    }
                    pc = pc -> next;
                }
                else if(strcmp(pc -> op,"rotl") == 0)
                {
                    if((temp=is_number(ptr_inarg -> next -> arg)) > INT_MIN)
                    {
                        if(_rotl_(value(pc -> procname,ptr_outarg ->
arg,0),value(pc -> procname,ptr_inarg -> arg,0),&temp) == 1);
                        else
                        {
                            fprintf(stderr,"\nError: Failure at statement
[rotl] at line: %ld.\n", _LINES_);
                            exit(-1);
                        }
                    }
                    else
                    {
                        if((_rotl_(value(pc -> procname,ptr_outarg ->
arg,0),value(pc -> procname,ptr_inarg -> arg,0),value(pc ->
procname,ptr_inarg -> next -> arg,0))) == 1);
```

```
                        else
                        {
                            fprintf(stderr,"\nError: Failure at statement
                                        [rotl] at line: %ld.\n", _LINES_);
                            exit(-1);
                        }
                    }
                    pc = pc -> next;
                }
                else if(strcmp(pc -> op,"rotr") == 0)
                {
                    if((temp=is_number(ptr_inarg -> next -> arg)) > INT_MIN)
                    {
                        if(_rotr_(value(pc -> procname,ptr_outarg ->
arg,0),value(pc -> procname,ptr_inarg -> arg,0),&temp) == 1);
                        else
                        {
                            fprintf(stderr,"\nError: Failure at statement
                                        [rotr] at line: %ld.\n", _LINES_);
                            exit(-1);
                        }
                    }
                    else
                    {
                        if((_rotr_(value(pc -> procname,ptr_outarg ->
arg,0),value(pc -> procname,ptr_inarg -> arg,0),value(pc ->
procname,ptr_inarg -> next -> arg,0))) == 1);
                        else
                        {
                            fprintf(stderr,"\nError: Failure at statement
                                        [rotr] at line: %ld.\n", _LINES_);
                            exit(-1);
                        }
                    }
                    pc = pc -> next;
                }
                else if(strcmp(pc -> op,"mul") == 0)
                {
                    if((temp=is_number(ptr_inarg -> next -> arg)) > INT_MIN)
                    {
                        if(_mul_(value(pc -> procname,ptr_outarg ->
arg,0),value(pc -> procname,ptr_inarg -> arg,0),&temp) == 1);
                        else
                        {
                            fprintf(stderr,"\nError: Failure at statement
                                        [mul] at line: %ld.\n", _LINES_);
                            exit(-1);
                        }
                    }
                    else
                    {
                        if((_mul_(value(pc -> procname,ptr_outarg ->
arg,0),value(pc -> procname,ptr_inarg -> arg,0),value(pc ->
procname,ptr_inarg -> next -> arg,0))) == 1);
                        else
                        {
                            fprintf(stderr,"\nError: Failure at statement
                                        [mul] at line: %ld.\n", _LINES_);
                            exit(-1);
```

```
                        }
                    }
                    pc = pc -> next;
                }
                else if(strcmp(pc -> op,"muxeq") == 0)
                {
                     if((temp = is_number(ptr_inarg -> next -> arg)) >
INT_MIN && (is_number(ptr_inarg -> next -> next -> next -> arg) == INT_MIN))
                     {
                        if(_muxeq_(value(pc -> procname,ptr_outarg ->
arg,0),value(pc -> procname,ptr_inarg -> arg,0),&temp,value(pc ->
procname,ptr_inarg -> next -> next -> arg,0),value(pc -> procname,ptr_inarg
-> next -> next -> next -> arg,0)) == 1);
                        else
                        {
                            fprintf(stderr,"\nError: Failure at statement
[muxeq] at line: %ld.\n", _LINES_);
                            exit(-1);
                        }
                     }
                     else if ((temp = is_number(ptr_inarg -> next -> next ->
next -> arg)) > INT_MIN && (is_number(ptr_inarg -> next -> arg)) == INT_MIN)
                     {
                        if(_muxeq_(value(pc -> procname,ptr_outarg ->
arg,0),value(pc -> procname,ptr_inarg -> arg,0),value(pc ->
procname,ptr_inarg -> next -> arg,0),value(pc -> procname,ptr_inarg -> next
-> next -> arg,0),&temp) == 1);
                        else
                        {
                            fprintf(stderr,"\nError: Failure at statement
                                       [muxeq] at line: %ld.\n", _LINES_);
                            exit(-1);
                        }
                     }
                     else if ((temp = is_number(ptr_inarg -> next -> next ->
next -> arg)) == INT_MIN && (temp = is_number(ptr_inarg -> next -> arg)) ==
INT_MIN)
                     {
                        if(_muxeq_(value(pc -> procname,ptr_outarg ->
arg,0),value(pc -> procname,ptr_inarg -> arg,0),value(pc ->
procname,ptr_inarg -> next -> arg,0),value(pc -> procname,ptr_inarg -> next
-> next -> arg,0),value(pc -> procname,ptr_inarg -> next -> next -> next ->
arg,0)) == 1);
                        else
                        {
                            fprintf(stderr,"\nError: Failure at statement
                                        [muxeq] at line: %ld.\n", _LINES_);
                            exit(-1);
                        }
                     }
                     else if(is_number(ptr_inarg->next->next->arg)>INT_MIN)
                     {
                          fprintf(stderr,"\nError: Failure at statement
[muxeq] at line: %ld. The third input argument [%s] must be a variable and
not a literal.\n", _LINES_,ptr_inarg -> next -> next -> arg );
                          exit(-1);
                     }
                     else
                     {
```

```
                        temp = is_number(ptr_inarg->next->arg);
                        temp2=is_number(ptr_inarg->next->next->next->arg);
                        if(_muxeq_(value(pc -> procname,ptr_outarg ->
arg,0),value(pc -> procname,ptr_inarg -> arg,0),&temp,value(pc ->
procname,ptr_inarg -> next -> next -> arg,0),&temp2) == 1);
                        else
                        {
                            fprintf(stderr,"\nError: Failure at statement
                                      [muxeq] at line: %ld.\n", _LINES_);
                            exit(-1);
                        }
                    }
                    pc = pc -> next;
                }
                else if(strcmp(pc -> op,"muxne") == 0)
                {
                    if((temp = is_number(ptr_inarg -> next -> arg)) >
INT_MIN && (is_number(ptr_inarg -> next -> next -> next -> arg) == INT_MIN))
                    {
                        if(_muxne_(value(pc -> procname,ptr_outarg ->
arg,0),value(pc -> procname,ptr_inarg -> arg,0),&temp,value(pc ->
procname,ptr_inarg -> next -> next -> arg,0),value(pc -> procname,ptr_inarg
-> next -> next -> next -> arg,0)) == 1);
                        else
                        {
                            fprintf(stderr,"\nError: Failure at statement
                                      [muxne] at line: %ld.\n", _LINES_);
                            exit(-1);
                        }
                    }
                    else if ((temp = is_number(ptr_inarg -> next -> next ->
next -> arg)) > INT_MIN && (temp = is_number(ptr_inarg -> next -> arg)) ==
INT_MIN)
                    {
                        if(_muxne_(value(pc -> procname,ptr_outarg ->
arg,0),value(pc -> procname,ptr_inarg -> arg,0),value(pc ->
procname,ptr_inarg -> next -> arg,0),value(pc -> procname,ptr_inarg -> next
-> next -> arg,0),&temp) == 1);
                        else
                        {
                            fprintf(stderr,"\nError: Failure at statement
                                      [muxne] at line: %ld.\n", _LINES_);
                            exit(-1);
                        }
                    }
                    else if ((temp = is_number(ptr_inarg -> next -> next ->
next -> arg)) == INT_MIN && (temp = is_number(ptr_inarg -> next -> arg)) ==
INT_MIN)
                    {
                        if(_muxne_(value(pc -> procname,ptr_outarg ->
arg,0),value(pc -> procname,ptr_inarg -> arg,0),value(pc ->
procname,ptr_inarg -> next -> arg,0),value(pc -> procname,ptr_inarg -> next
-> next -> arg,0),value(pc -> procname,ptr_inarg -> next -> next -> next ->
arg,0)) == 1);
                        else
                        {
                            fprintf(stderr,"\nError: Failure at statement
                                      [muxne] at line: %ld.\n", _LINES_);
                            exit(-1);
```

```
                        }
                    }
                    else if(is_number(ptr_inarg->next->next->arg)>INT_MIN)
                    {
                         fprintf(stderr,"\nError: Failure at statement
[muxne] at line: %ld. The third input argument [%s] must be a variable and
not a literal.\n", _LINES_,ptr_inarg -> next -> next -> arg );
                         exit(-1);
                    }
                    else
                    {
                        temp = is_number(ptr_inarg->next->arg);
                        temp2=is_number(ptr_inarg->next->next->next->arg);
                        if(_muxne_(value(pc -> procname,ptr_outarg ->
arg,0),value(pc -> procname,ptr_inarg -> arg,0),&temp,value(pc ->
procname,ptr_inarg -> next -> next -> arg,0),&temp2) == 1);
                        else
                        {
                            fprintf(stderr,"\nError: Failure at statement
                                        [muxne] at line: %ld.\n", _LINES_);
                            exit(-1);
                        }
                    }
                    pc = pc -> next;
                }
                else if(strcmp(pc -> op,"muxlt") == 0)
                {
                    if((temp = is_number(ptr_inarg -> next -> arg)) >
INT_MIN && (is_number(ptr_inarg -> next -> next -> next -> arg) == INT_MIN))
                    {
                        if(_muxlt_(value(pc -> procname,ptr_outarg ->
arg,0),value(pc -> procname,ptr_inarg -> arg,0),&temp,value(pc ->
procname,ptr_inarg -> next -> next -> arg,0),value(pc -> procname,ptr_inarg
-> next -> next -> next -> arg,0)) == 1);
                        else
                        {
                            fprintf(stderr,"\nError: Failure at statement
                                         [muxlt] at line: %ld.\n", _LINES_);
                            exit(-1);
                        }
                    }
                    else if ((temp = is_number(ptr_inarg -> next -> next ->
next -> arg)) > INT_MIN && (temp = is_number(ptr_inarg -> next -> arg)) ==
INT_MIN)
                    {
                        if(_muxlt_(value(pc -> procname,ptr_outarg ->
arg,0),value(pc -> procname,ptr_inarg -> arg,0),value(pc ->
procname,ptr_inarg -> next -> arg,0),value(pc -> procname,ptr_inarg -> next
-> next -> arg,0),&temp) == 1);
                        else
                        {
                            fprintf(stderr,"\nError: Failure at statement
                                        [muxlt] at line: %ld.\n", _LINES_);
                            exit(-1);
                        }
                    }
                    else if ((temp = is_number(ptr_inarg -> next -> next ->
next -> arg)) == INT_MIN && (temp = is_number(ptr_inarg -> next -> arg)) ==
INT_MIN)
```

```
                    {
                       if(_muxlt_(value(pc -> procname,ptr_outarg ->
arg,0),value(pc -> procname,ptr_inarg -> arg,0),value(pc ->
procname,ptr_inarg -> next -> arg,0),value(pc -> procname,ptr_inarg -> next
-> next -> arg,0),value(pc -> procname,ptr_inarg -> next -> next -> next ->
arg,0)) == 1);
                       else
                       {
                           fprintf(stderr,"\nError: Failure at statement
                                       [muxlt] at line: %ld.\n", _LINES_);
                           exit(-1);
                       }
                    }
                    else if(is_number(ptr_inarg->next->next->arg)>INT_MIN)
                    {
                         fprintf(stderr,"\nError: Failure at statement
[muxlt] at line: %ld. The third input argument [%s] must be a variable and
not a literal.\n", _LINES_,ptr_inarg -> next -> next -> arg );
                         exit(-1);
                    }
                    else
                    {
                        temp = is_number(ptr_inarg->next->arg);
                        temp2=is_number(ptr_inarg->next->next->next->arg);
                        if(_muxlt_(value(pc -> procname,ptr_outarg ->
arg,0),value(pc -> procname,ptr_inarg -> arg,0),&temp,value(pc ->
procname,ptr_inarg -> next -> next -> arg,0),&temp2) == 1);
                       else
                       {
                           fprintf(stderr,"\nError: Failure at statement
                                       [muxlt] at line: %ld.\n", _LINES_);
                           exit(-1);
                       }
                    }
                    pc = pc -> next;
                }
                else if(strcmp(pc -> op,"muxle") == 0)
                {
                    if((temp = is_number(ptr_inarg -> next -> arg)) >
INT_MIN && (is_number(ptr_inarg -> next -> next -> next -> arg) == INT_MIN))
                    {
                        if(_muxle_(value(pc -> procname,ptr_outarg ->
arg,0),value(pc -> procname,ptr_inarg -> arg,0),&temp,value(pc ->
procname,ptr_inarg -> next -> next -> arg,0),value(pc -> procname,ptr_inarg
-> next -> next -> next -> arg,0)) == 1);
                       else
                       {
                           fprintf(stderr,"\nError: Failure at statement
                                       [muxle] at line: %ld.\n", _LINES_);
                           exit(-1);
                       }
                    }
                    else if ((temp = is_number(ptr_inarg -> next -> next ->
next -> arg)) > INT_MIN && (temp = is_number(ptr_inarg -> next -> arg)) ==
INT_MIN)
                    {
                       if(_muxle_(value(pc -> procname,ptr_outarg ->
arg,0),value(pc -> procname,ptr_inarg -> arg,0),value(pc ->
```

```
procname,ptr_inarg -> next -> arg,0),value(pc -> procname,ptr_inarg -> next
-> next -> arg,0),&temp) == 1);
                        else
                        {
                            fprintf(stderr,"\nError: Failure at statement
                                        [muxle] at line: %ld.\n", _LINES_);
                            exit(-1);
                        }
                    }
                    else if ((temp = is_number(ptr_inarg -> next -> next ->
next -> arg)) == INT_MIN && (temp = is_number(ptr_inarg -> next -> arg)) ==
INT_MIN)
                    {
                        if(_muxle_(value(pc -> procname,ptr_outarg ->
arg,0),value(pc -> procname,ptr_inarg -> arg,0),value(pc ->
procname,ptr_inarg -> next -> arg,0),value(pc -> procname,ptr_inarg -> next
-> next -> arg,0),value(pc -> procname,ptr_inarg -> next -> next -> next ->
arg,0)) == 1);
                        else
                        {
                            fprintf(stderr,"\nError: Failure at statement
                                        [muxle] at line: %ld.\n", _LINES_);
                            exit(-1);
                        }
                    }
                    else if(is_number(ptr_inarg->next->next->arg)>INT_MIN)
                    {
                         fprintf(stderr,"\nError: Failure at statement
[muxle] at line: %ld. The third input argument [%s] must be a variable and
not a literal.\n", _LINES_,ptr_inarg -> next -> next -> arg );
                         exit(-1);
                    }
                    else
                    {
                        temp = is_number(ptr_inarg->next->arg);
                        temp2 = is_number(ptr_inarg->next->next->next-
                                                                >arg);
                        if(_muxle_(value(pc -> procname,ptr_outarg ->
arg,0),value(pc -> procname,ptr_inarg -> arg,0),&temp,value(pc ->
procname,ptr_inarg -> next -> next -> arg,0),&temp2) == 1);
                        else
                        {
                            fprintf(stderr,"\nError: Failure at statement
                                        [muxle] at line: %ld.\n", _LINES_);
                            exit(-1);
                        }
                    }
                    pc = pc -> next;
                }
                else if(strcmp(pc -> op,"muxgt") == 0)
                {
                    if((temp = is_number(ptr_inarg -> next -> arg)) >
INT_MIN && (is_number(ptr_inarg -> next -> next -> next -> arg) == INT_MIN))
                    {
                        if(_muxgt_(value(pc -> procname,ptr_outarg ->
arg,0),value(pc -> procname,ptr_inarg -> arg,0),&temp,value(pc ->
procname,ptr_inarg -> next -> next -> arg,0),value(pc -> procname,ptr_inarg
-> next -> next -> next -> arg,0)) == 1);
                        else
```

```
                        {
                            fprintf(stderr,"\nError: Failure at statement
                                        [muxgt] at line: %ld.\n", _LINES_);
                            exit(-1);
                        }
                    }
                    else if ((temp = is_number(ptr_inarg -> next -> next ->
next -> arg)) > INT_MIN && (temp = is_number(ptr_inarg -> next -> arg)) ==
INT_MIN)
                    {
                        if(_muxgt_(value(pc -> procname,ptr_outarg ->
arg,0),value(pc -> procname,ptr_inarg -> arg,0),value(pc ->
procname,ptr_inarg -> next -> arg,0),value(pc -> procname,ptr_inarg -> next
-> next -> arg,0),&temp) == 1);
                        else
                        {
                            fprintf(stderr,"\nError: Failure at statement
                                        [muxgt] at line: %ld.\n", _LINES_);
                            exit(-1);
                        }
                    }
                    else if ((temp = is_number(ptr_inarg -> next -> next ->
next -> arg)) == INT_MIN && (temp = is_number(ptr_inarg -> next -> arg)) ==
INT_MIN)
                    {
                        if(_muxgt_(value(pc -> procname,ptr_outarg ->
arg,0),value(pc -> procname,ptr_inarg -> arg,0),value(pc ->
procname,ptr_inarg -> next -> arg,0),value(pc -> procname,ptr_inarg -> next
-> next -> arg,0),value(pc -> procname,ptr_inarg -> next -> next -> next ->
arg,0)) == 1);
                        else
                        {
                            fprintf(stderr,"\nError: Failure at statement
                                        [muxgt] at line: %ld.\n", _LINES_);
                            exit(-1);
                        }
                    }
                    else if(is_number(ptr_inarg->next->next->arg)>INT_MIN)
                    {
                         fprintf(stderr,"\nError: Failure at statement
[muxgt] at line: %ld. The third input argument [%s] must be a variable and
not a literal.\n", _LINES_,ptr_inarg -> next -> next -> arg );
                         exit(-1);
                    }
                    else
                    {
                        temp = is_number(ptr_inarg->next->arg);
                        temp2 = is_number(ptr_inarg->next->next->next-
                                                                >arg);
                        if(_muxgt_(value(pc -> procname,ptr_outarg ->
arg,0),value(pc -> procname,ptr_inarg -> arg,0),&temp,value(pc ->
procname,ptr_inarg -> next -> next -> arg,0),&temp2) == 1);
                        else
                        {
                            fprintf(stderr,"\nError: Failure at statement
[muxgt] at line: %ld.\n", _LINES_);
                            exit(-1);
                        }
                    }
```

```
                    pc = pc -> next;
                }
                else if(strcmp(pc -> op,"muxge") == 0)
                {
                    if((temp = is_number(ptr_inarg -> next -> arg)) >
INT_MIN && (is_number(ptr_inarg -> next -> next -> next -> arg) == INT_MIN))
                    {
                        if(_muxge_(value(pc -> procname,ptr_outarg ->
arg,0),value(pc -> procname,ptr_inarg -> arg,0),&temp,value(pc ->
procname,ptr_inarg -> next -> next -> arg,0),value(pc -> procname,ptr_inarg
-> next -> next -> next -> arg,0)) == 1);
                        else
                        {
                            fprintf(stderr,"\nError: Failure at statement
                                        [muxge] at line: %ld.\n", _LINES_);
                            exit(-1);
                        }
                    }
                    else if ((temp = is_number(ptr_inarg -> next -> next ->
next -> arg)) > INT_MIN && (temp = is_number(ptr_inarg -> next -> arg)) ==
INT_MIN)
                    {
                        if(_muxge_(value(pc -> procname,ptr_outarg ->
arg,0),value(pc -> procname,ptr_inarg -> arg,0),value(pc ->
procname,ptr_inarg -> next -> arg,0),value(pc -> procname,ptr_inarg -> next
-> next -> arg,0),&temp) == 1);
                        else
                        {
                            fprintf(stderr,"\nError: Failure at statement
                                       [muxge] at line: %ld.\n", _LINES_);
                            exit(-1);
                        }
                    }
                    else if ((temp = is_number(ptr_inarg -> next -> next ->
next -> arg)) == INT_MIN && (temp = is_number(ptr_inarg -> next -> arg)) ==
INT_MIN)
                    {
                        if(_muxge_(value(pc -> procname,ptr_outarg ->
arg,0),value(pc -> procname,ptr_inarg -> arg,0),value(pc ->
procname,ptr_inarg -> next -> arg,0),value(pc -> procname,ptr_inarg -> next
-> next -> arg,0),value(pc -> procname,ptr_inarg -> next -> next -> next ->
arg,0)) == 1);
                        else
                        {
                            fprintf(stderr,"\nError: Failure at statement
                                        [muxge] at line: %ld.\n", _LINES_);
                            exit(-1);
                        }
                    }
                    else if(is_number(ptr_inarg->next->next->arg)>INT_MIN)
                    {
                         fprintf(stderr,"\nError: Failure at statement
[muxge] at line: %ld. The third input argument [%s] must be a variable and
not a literal.\n", _LINES_,ptr_inarg -> next -> next -> arg );
                         exit(-1);
                    }
                    else
                    {
                        temp = is_number(ptr_inarg->next->arg);
```

```
                        temp2 = is_number(ptr_inarg->next->next->next-
                                                                >arg);
                        if(_muxge_(value(pc -> procname,ptr_outarg ->
arg,0),value(pc -> procname,ptr_inarg -> arg,0),&temp,value(pc ->
procname,ptr_inarg -> next -> next -> arg,0),&temp2) == 1);
                      else
                      {
                          fprintf(stderr,"\nError: Failure at statement
                                      [muxge] at line: %ld.\n", _LINES_);
                          exit(-1);
                      }
                    }
                    pc = pc -> next;
                }
            }
            else //Procedure call
            {
                checks_if_proc_exists(pc -> op);
                pla = find_address(pc -> op);
                copy_proc_in_args(pc -> procname, pc -> op, pc -> in);
                do_call(pc, pla -> address, pc -> procname, pc -> op);
                pc = pla -> address;
             }
        }

       if ((pass_by_value_out(fh -> out,times)) != 0)
       {
            fprintf(stderr,"\n Unable to pass the output values from file:
                                              [%s].\n", INPUT_DATA_FNAME);
            exit(-1);
       }

        times++;
        fh = fh -> next;
    }
    return 0;
}

// print_localvar_for_list
int print_localvar_for_list()
{
    range_loc *tmp_range = range_head;

    printf("\n______PROCEDURE ARGUMENTS & LOCAL VARIABLES______");
    if(tmp_range != NULL)
    {
        printf("\n\nProcedure Name: [%s]", tmp_range -> procname);
        printf("\nProcedure InOut Arguments: -> \n\n");
        print_localvar_list(tmp_range -> lv_list, tmp_range -> pinarg,
                                              tmp_range -> poutarg);
        while(tmp_range -> next != NULL)
        {
                tmp_range = tmp_range -> next;

                printf("\n\nProcedure Name: [%s]", tmp_range -> procname);
                printf("\nProcedure InOut Arguments: -> \n\n");
                print_localvar_list(tmp_range -> lv_list, tmp_range ->
                                        pinarg, tmp_range -> poutarg);
        }
```

```
    }
    return 0;
}

int print_in_out_for_stack(inarg *inhead, outarg *outhead)
{
    inarg *tmpin = inhead;
    outarg *tmpout = outhead;

    if(tmpin != NULL)
    {
        while(1)
        {
            if((tmpin -> next == NULL) || (tmpin == intail))
            {
                printf("[%s]", tmpin -> arg);
                break;
            }
            else
                 printf("[%s] - ", tmpin -> arg);

            tmpin = tmpin -> next;
        }
    }
    if(tmpout != NULL)
    {
        while(1)
        {
            if((tmpout -> next == NULL) || (tmpout == outail))
            {
                printf("[%s]", tmpout -> arg);
                break;
            }
            else
                 printf("[%s] - ", tmpout -> arg);

            tmpout = tmpout -> next;
        }
    }
    return 0;
}

int print_index_value_gv(init_array *arhead)
{
    init_array *tmp = arhead;

    if(tmp != NULL)
    {
        while(1)
        {
            if(tmp -> next == NULL)
            {
                printf("[%ld]", tmp -> index);
                printf(" - [%d]\n", tmp -> value);
                break;
            }
            else
                 printf("[%ld]", tmp -> index);
                 printf(" - [%d]\n", tmp -> value);
```

```
            tmp = tmp -> next;
        }
    }
    return 0;
 }

int print_index_value_lv(init_array *arlochead)
{
    init_array *tmp = arlochead;

    if(tmp != NULL)
    {
        while(1)
        {
            if(tmp -> next == NULL)
            {
                printf("[%ld]", tmp -> index);
                printf(" - [%d]\n", tmp -> value);
                break;
            }
            else
                 printf("[%ld]", tmp -> index);
                 printf(" - [%d]\n", tmp -> value);

            tmp = tmp -> next;
        }
    }
    return 0;
 }

// print_globalvar
void print_globalvar(void)
{
    globalvar_list *tmp = ghead;

    printf("\n______GLOBALVAR______\n\n");
    if(tmp == NULL)
    {
        fprintf(stderr,"\nEmpty GlobalVar.\n");
        terminate();
    }
    else
    {
        while(1)
        {
            printf("Node:      [%ld]\nVarName:   [%s]\nVarType:
[%s]\nArraySize: [%ld]",tmp -> gv_rank, tmp -> varname, tmp -> vartype, tmp
-> array_size);
            printf("\nIndex - Value\n");
            print_index_value_gv(tmp -> array);
            if(tmp -> next == NULL)
            {
                break;
            }
            else
            {
                tmp = tmp -> next;
                printf("\n\n");
```

```
            }
        }
        printf("\n\n");
    }
}

void print_quick_access_list (void)
{
    proc_lab *proc_tmp;
    label_address *la_tmp;

    proc_tmp = pl_head;

    while(1)
    {
        if (proc_tmp == NULL)
            break;

       la_tmp = proc_tmp -> la;
        while(1)
        {
            if(la_tmp == NULL)
                break;
            printf("\nProcedure: [%s]\nLabel: [%s]\nNode: [%d]\n", proc_tmp
-> proc_name, la_tmp -> label_name, la_tmp ->address->rank);
            la_tmp = la_tmp -> next;
        }
       printf("\n\n");
        proc_tmp = proc_tmp -> next;
    }
}

// print_localvar_list
int print_localvar_list(localvar_list *lhead, proc_inarg *proc_inhead,
proc_outarg *proc_outhead)
{
    localvar_list *tmp = lhead;
    proc_inarg *tmp_in = proc_inhead;
    proc_outarg *tmp_out = proc_outhead;

    if(tmp_in != NULL)
    {
        while(1)
        {
            printf("Node-In-Args: [%ld]\nVarName:      [%s]\nVarType:
[%s]\nArraySize:    [%ld]",tmp_in -> in_rank, tmp_in -> varname, tmp_in ->
vartype, tmp_in -> array_size);
            printf("\nIndex - Value\n");

            print_index_value_lv(tmp_in -> array);
            if(tmp_in -> next == NULL)
            {
                printf("\n");
                break;
            }
            else
            {
                tmp_in = tmp_in -> next;
                printf("\n");
```

```
            }
        }
    }

    if(tmp_out != NULL)
    {
        while(1)
        {
            printf("Node-Out-Args: [%ld]\nVarName:       [%s]\nVarType:
[%s]\nArraySize:     [%ld]",tmp_out -> out_rank, tmp_out -> varname, tmp_out
-> vartype, tmp_out -> array_size);
            printf("\nIndex - Value\n");
            print_index_value_lv(tmp_out -> array);
            if(tmp_out -> next == NULL)
            {
                printf("\n");
                break;
            }
            else
            {
                tmp_out = tmp_out -> next;
                printf("\n");
            }
        }
    }

    if(tmp_in == NULL && tmp_out == NULL)
        printf("\nProcedure without in-out arguments.\n");

    if(tmp == NULL)
    {
        fprintf(stderr,"\nEmpty LocalVar.\n");
    }
    else
    {
        printf("_________________\nLocal Variables: ->\n_________________\n\n");
        while(1)
        {
            printf("Node:      [%ld]\nVarName:   [%s]\nVarType:
[%s]\nArraySize: [%ld]",tmp -> lv_rank, tmp -> varname, tmp -> vartype, tmp
-> array_size);
            printf("\nIndex - Value\n");
            print_index_value_lv(tmp -> array);
            if(tmp -> next == NULL)
            {
                break;
            }
            else
            {
                tmp = tmp -> next;
                printf("\n");
            }
        }
    }
    return 0;
}

void print_localvar(void)
{
```

```
    localvar_list *tmp = lhead;

    printf("\n______LOCALVAR______\n\n");
    if(tmp == NULL)
    {
        fprintf(stderr,"\nEmpty LocalVar.\n");
        terminate();
    }
    else
    {
        while(1)
        {
            printf("Node:      [%ld]\nVarName:   [%s]\nVarType:
[%s]\nArraySize: [%ld]",tmp -> lv_rank, tmp -> varname, tmp -> vartype, tmp
-> array_size);
            printf("\nIndex - Value\n");
            print_index_value_lv(tmp -> array);
            if(tmp -> next == NULL)
            {
                break;
            }
            else
            {
                tmp = tmp -> next;
                printf("\n\n");
            }
        }
        printf("\n\n");
    }
}

// print_stack
void print_stack(void)
{
    instruction *tmp = head;

    printf("\n______STACK______\n\n");
    if(tmp == NULL)
    {
        fprintf(stderr,"\nEmpty Stack.\n");
        terminate();
    }
    else
    {
        while(1)
        {
            printf("Node:      [%d]\nProcedure: [%s]\nPCALL:
[%c]\nLabel:     [%s]\nOperation: [%s]", tmp -> rank, tmp -> procname, tmp -
> pcall,tmp -> label,tmp -> op);
            printf("\nOut Args:  ");
            print_in_out_for_stack(NULL,tmp -> out);
            printf("\nIn Args:   ");
            print_in_out_for_stack(tmp -> in,NULL);
            if(tmp -> next == NULL)
            {
                break;
            }
            else
            {
```

```
                tmp = tmp -> next;
                printf("\n\n");
            }
        }
        printf("\n\n");
    }
}

void dummy_procedure(void)
{
    instruction *ptr;
    create_lv_range_list("dummy_procedure_!@#6891","no");

    ptr = (instruction *)malloc(sizeof(instruction));
    if (ptr == NULL)
    {
            fprintf(stderr,"\nError: Memory Allocation: At function
                                                dummy_procedure()\n");
            exit(-1);
    }

    ptr -> op = (char *)malloc(sizeof("nop"));
    if(ptr -> op == NULL)
    {
        fprintf(stderr,"\nError: Memory Allocation: At function
                                            dummy_procedure().\n");
        exit(-1);
    }
    strcpy(ptr -> op, "nop");

    ptr -> in = NULL;
    ptr -> out = NULL;

    ptr -> procname = (char *)malloc(sizeof("dummy_procedure_!@#6891"));
    if(ptr -> procname == NULL)
    {
        fprintf(stderr,"\nError: Memory Allocation: At function
                                                  dummy_procedure().\n");
        exit(-1);
    }
    strcpy(ptr -> procname, "dummy_procedure_!@#6891");

    ptr -> label = (char *)malloc(sizeof("dummy_label_!@#6891"));
    if(ptr -> label == NULL)
    {
        fprintf(stderr,"\nError: Memory Allocation: At function
                                                dummy_procedure().\n");
        exit(-1);
    }
    strcpy(ptr -> label, "dummy_label_!@#6891");

    ptr -> pcall = '0';
    ptr -> rank = rank;

    tail -> next = ptr;
    ptr -> previous = tail;
    tail = ptr;
    tail -> next = NULL;
}
```