ΜΕΤΑΦΡΑΣΤΗΣ MINI PASCAL
ΤΕΛΙΚΗ ΑΝΑΦΟΡΑ

Περιεχόμενα:

1. Εισαγωγή

Δημιουργήσαμε έναν μεταφραστή ο οποίος λαμβάνει ως εισαγωγή ένα πρόγραμμα γραμμένο σε miniPascal και το μεταφράζει σε γλώσσα μηχανής (assembly). Χρησιμοποιήσαμε γλώσσα C++ (c++11) αλλά και πολλές συναρτήσεις από τη C. Η μετάφραση υλοποιείται σε 5 φάσεις.

- Ο λεκτικός αναλυτής λαμβάνει το αρχείο προγράμματος σε miniPascal και το μετατρέπει σε ένα σύνολο λεκτικών μονάδων, τις οποίες διαχειρίζεται ο συντακτικός αναλυτής.

- Ο συντακτικός αναλυτής λαμβάνεις τις λεκτικές μονάδες από τον λεκτικό αναλυτής και βλέπει εάν συμφωνούν με τη σύνταξη/γραμματική της miniPascal. Εάν δε συμφωνούν βγάζει διαγνωστικά μηνύματα στα οποία αναφέρεται η ασυμφωνία της λεκτικής μονάδας με την γραμματική.

- Μετά την συντακτική ανάλυση παράγεται ο ενδιάμεσος κώδικας. Ο ενδιάμεσος κώδικας είναι ένα σύνολο από αριθμημένες τετράδες αλφαριθμητικών οι οποίες βοηθούν στην παραγωγή του τελικού κώδικα.

- Δημιουργείται, ύστερα, ο πίνακας συμβόλων ως βοηθητικό εργαλείο για την δημιουργία του εγγραφήματος δραστηριοποίησης και την παραγωγή του τελικού κώδικα

- Τέλος, με βάση τις τετράδες του ενδιάμεσου κώδικα και τον πίνακα συμβόλων, παράγεται ο τελικός κώδικας του προγράμματος σε γλώσσα μηχανής.

2. Γλώσσα Mini Pascal

Η γλώσσα mini Pascal είναι μία απλουστευμένη υλοποίηση της Pascal. Υποστηρίζει συναρτήσεις, διαδικασίες, μετάδοση παραμέτρων με αναφορά και τιμή, αναδρομικές κλήσεις, εμφώλιασμα συναρτήσεων και τις περισσότερες γνωστές δομές ελέγχου. Επίσης υποστηρίζει μόνο ακεραίους (δεν υποστηρίζει πραγματικούς αριθμούς, χαρακτήρες, συμβολοσειρές και πίνακες ή αλλες δομές δεδομένων).

Το αλφάβητο της Mini_Pascal αποτελείται από:

- τα μικρά και κεφαλαία γράμματα του λατινικού αλφαβήτου
- τα αριθμητικά ψηφία { 0 1 2 3 4 5 6 7 8 9 }
- τα σύμβλα των αριθμητικών πράξεων { + - * / }
- τους τελεστές συσχέτισης { < > <= >= = <> }
- το σύμβολο ανάθεσης { := }
- τους διαχωριστές { ; , }
- τα σύμβολα ομαδοποίησης { ( ) [ ] }
- τα σύμβολα διαχωρισμού σχολίων { /* */ }
- τις δεσμευμένες λέξεις { program begin end const var procedure function if then else while do for to step call print input or and not return select endselect }

Το συντακτικό της Mini_Pascal που χρησιμοποιήσαμε είναι το εξής:

```
<PROGRAM>           ::=   "program" ID <PROGRAMBLOCK>

<PROGRAMBLOCK>      ::=   <DECLARATIONS> <SUBPROGRAMS> <BLOCK>

<BLOCK>             ::=   "begin" <SEQUENCE> "end"

<DECLARATIONS>      ::=   ( <CONSTDECL> )* ( <VARDECL> )*

<CONSTDECL>         ::=   "const" <ASSIGNLIST> ";" |        ε

<ASSIGNLIST>        ::=   ASSIGNCONST ( , ASSIGNCONST )*

<ASSIGNCONST>       ::=   ID ":=" CONSTANT

<VARDECL>           ::=   "var" <VARLIST> ";" |     ε

<VARLIST>           ::=   ID ( , ID )*

<SUBPROGRAMS>       ::=   ( <PROCORFUNC> )*

<PROCORFUNC>        ::=   "procedure" ID <PROCORFUNCBODY> |
                          "function" ID <PROCORFUNCBODY>

<PROCORFUNCBODY>    ::=   <FORMALPARS> <PROGRAMBLOCK>

<FORMALPARS>        ::=   "(" <FORMALPARLIST> | ε ")"

<FORMALPARLIST>     ::=   <FORMALPARITEM> ( , <FORMALPARITEM> )*

<FORMALPARITEM>     ::=   ID |
                          "var" ID

<SEQUENCE>          ::=   <STATEMENT> ( ";" <STATEMENT> )*

<BLOCK-OR-STAT>     ::=    <BLOCK> |
                           <STATEMENT>

<STATEMENT>         ::=   <ASSIGNMENT-STAT> |
                          <IF-STAT> |
```

```
                                <WHILE-STAT> |
                                <FOR-STAT> |
                                <CALL-STAT> |
                                <PRINT-STAT> |
                                <INPUT-STAT> |
                                <RETURN-STAT>|
                                <SELECT-STAT>

<ASSIGNMENT-STAT>     ::=       ID  ":="  <EXPRESSION>

<IF-STAT>             ::=       "if"  <CONDITION>  "then"  <BLOCK-OR-STAT>  <ELSEPART>

<ELSEPART>            ::=       ε  |         "else" <BLOCK-OR-STAT>

<WHILE-STAT>          ::=       "while"  <CONDITION>  "do"  <BLOCK-OR-STAT>

<FOR-STAT>            ::=       "for"  <ASSIGNMENT-STAT>  "to"  <EXPRESSION>  <STEP-PART>  <BLOCK-OR-STAT>

<STEP-PART>           ::=       "step"  <EXPRESSION>  |   ε

<CALL-STAT>           ::=       "call"  ID  <ACTUALPARS>

<PRINT-STAT>          ::=       "print" "(" <EXPRESSION> ")"

<INPUT-STAT>          ::=       "input" "(" ID ")"

<RETURN-STAT>         ::=       "return" "(" <EXPRESSION> ")"

<SELECT-STAT>         ::=        "select" ID  <IS-EQUAL-PART> ( <IS-EQUAL-PART> )* "endselect"

<IS-EQUAL-PART>       ::=       "is equal to" <EXPRESSION> ":" <BLOCK>

<ACTUALPARS>          ::=       "(" <ACTUALPARLIST> | ε  ")"

<ACTUALPARLIST>       ::=       <ACTUALPARITEM> ( , <ACTUALPARITEM> )*

<ACTUALPARITEM>       ::=       <EXPRESSION> |
                                "var" ID

<CONDITION>           ::=       <BOOLTERM> ( "or" <BOOLTERM>)*

<BOOLTERM>            ::=       <BOOLFACTOR> ( "and"  <BOOLFACTOR> )*

<BOOLFACTOR>          ::=       "not" "[" <CONDITION> "]" |
                                "[" <CONDITION> "]" |
                                <EXPRESSION>  <RELATIONAL-OPER>  <EXPRESSION>

<RELATIONAL-OPER>     ::=        "=" |
                                "<" |
                                ">" |
                                ">=" |
                                "<=" |
                                "<>"

<EXPRESSION>          :=        <OPTIONAL-SIGN>  <TERM>  ( <ADD-OPER> <TERM>)*

<TERM>                ::=       <FACTOR> (<MUL-OPER> <FACTOR>)*

<FACTOR>              ::=       CONSTANT |
                                "(" <EXPRESSION> ")"  |
                                ID  <OPTACTUALPARS>

<OPTACTUALPARS>       ::=       <ACTUALPARS> | ε

<ADD-OPER>            ::=        "+"  |  "-"

<MUL-OPER>            ::=       "*"  |  "/"

<OPTIONAL-SIGN>       ::=       ε  |  <ADD-OPER>
```

3. Λεκτικός Αναλυτής

Ο Λεκτικός Αναλυτής καλείται από τον συντακτικό αναλυτή. Σε κάθε κλήση διαβάζει από το αρχείο που περιέχει τον κώδικα προς μετάφραση και επιστρέφει, στο συντακτικό αναλυτή, την επόμενη λεκτική μονάδα και έναν ακέραιο που χαρακτηρίζει την λεκτική μονάδα. Επίσης ο Λεκτικός αναλυτής αφαιρεί τα σχόλια.

Ο Λεκτικός Αναλυτής αναγνωρίζει ως λεκτικές μονάδες τα εξής:

- τις δεσμευμένες λέξεις { program begin end const var procedure function if then else while do for to step call print input or and not return select endselect }
- τα σύμβολα της γλώσσας { ; , ( ) [ ] + - * / < > = <= >= <>}
- αριθμητικές σταθερές (φυσικούς αριθμούς με |αριθμός|<=2147483647)
- αναγνωριστικά (ονόματα μεταβλητών, διεργασιών, συναρτήσεων κλπ)

Μεταβλητές, δομές δεδομένων και σταθερές

```
#define     idtk    100   //variable or function name (ID)
#define     numtk   101   //number (CONSTANT)
#define     addtk   200   // +  addition
#define     subtk   201   // -  subtraction
#define     multk   202   // *  multiplication
#define     divtk   203   // /  division
#define     smatk   204   // <  smaller than
#define     bigtk   205   // >  bigger than
#define     equtk   206   // =  equal to
#define     soetk   207   // <= smaller or equal
#define     boetk   208   // >= bigger or equal
#define     diftk   209   // <> different than
#define     asstk   210   // := assign value
#define     semtk   300   // ;  semicolon
#define     comtk   301   // ,  comma
#define     paotk   302   // (  parentheses open
#define     pactk   303   // )  parentheses close
#define     sbotk   304   // [  square bracket open
#define     sbctk   305   // ]  square bracket close
#define     coltk   306   // :  colon
#define     progtk  400   // program
#define     begitk  401   // begin
#define     endxtk  402   // end
#define     constk  403   // const
#define     varxtk  404   // var
#define     proctk  405   // procedure
#define     functk  406   // function
#define     ifxxtk  407   // if
#define     thentk  408   // then
#define     elsetk  409   // else
#define     whiltk  410   // while
#define     doxxtk  411   // do
#define     forxtk  412   // for
#define     toxxtk  413   // to
#define     steptk  414   // step
#define     calltk  415   // call
```

```
#define     printk  416     // print
#define     inputk 417      // input
#define     orxxtk 418      // or
#define     andxtk 419      // and
#define     notxtk 420      // not
#define     retutk  421     // return
#define     seletk  422     // select
#define     enseltk         423     // endselect
#define     eoftk   001     //eof
#define     errtk   000     //error

FILE *finp;
int lineCount;
const string reswords_array[] =
{"program","begin","end","const","var","procedure","function","if","then","else","while","do","for","to","step
","call","print","input","or","and","not","return","select", "endselect"};
const string symb = ";,()[]";
const string oper = "+-*/=<>:";
int tokenid;
string token;
```

Συναρτήσεις

void spaceEater() — Καλείται από την lex() με σκοπό το προσπέρασμα των κενών διαστημάτων και των σχολίων του εισαγόμενου προγράμματος και τοποθέτηση του file pointer στην επόμενη λεκτική μονάδα

int checkNumValidity() — Καλείται από τη lex() για τον έλεγχο του μεγέθους ενός αριθμού. Η miniPascal επιτρέπει αριθμούς μικρότερους, κατά απόλυτη τιμή, από 2147483647.

int lex() — Η κύρια συνάρτηση του Λεκτικού Αναλυτή. Διαβάζει χαρακτήρα-χαρακτήρα το αρχείο του αρχικού κώδικα και βρίσκει την επόμενη λεκτική μονάδα, την τοποθετεί στη μεταβλητή token και επιστρέφει τη τιμή της σταθεράς που χαρακτηρίζει αυτή τη λεκτική μονάδα.
Πρώτα καλεί την spaceEater() για να προσπεράσει τα κενά και τα σχόλια. Μετά διαβάζει τον πρώτο χαρακτήρα της λεκτικής μονάδας.
Αν ο χαρακτήρας είναι γράμμα, συμπεραίνει ότι η λεκτική μονάδα είναι αναγνωριστικό ή δεσμευμένη λέξη. Διαβάζει ολόκληρη τη λεκτική μονάδα και τη συγκρίνει με τις δεσμευμένες λέξεις. Αν είναι δεσμευμένη λέξη ή αναγνωριστικό επιστρέφει την ανάλογη σταθερά.
Αν ο χαρακτήρας είναι αριθμός, διαβάζει όλο τον αριθμό και ελέγχει, καλώντας την checkNumValidity() αν είναι εντός ορίων.
Αν ο χαρακτήρας είναι ένας από τα σύμβολα ; , ( ) [ ] επιστρέφει την ανάλογη σταθερά.
Αν ο χαρακτήρας είναι κάποιος από τους + - * / = < > : ελέγχει αν βρήκε κάποιο άλλο σύμβολο της γλώσσας και επιστρέφει την ανάλογη σταθερά.

4. Συντακτικός Αναλυτής

Ο Συντακτικός Αναλυτής είναι ένα σύνολο από συναρτήσεις οι οποίες ελέγχουν αν το πρόγραμμα προς μετάφραση συμφωνεί με τους κανόνες της γραμματικής (σελ 3 & 4). Οι συναρτήσεις του συντακτικού αναλυτή αναπαριστούν και από ένα κανόνα της γραμματικής. Κάποιες συναρτήσεις ενσωματώνουν και πάνω από ένα κανόνα με αποτέλεσμα να μην είναι απαραίτητη η υλοποίηση των κανόνων <RELATIONAL-OPER>, <OPTACTUALPARS>, <ADD-OPER>, <MUL-OPER>, <OPTIONAL-SIGN>. Η αρχική συνάρτηση που καλείται στην main και εκκινεί τον συντακτικό αναλυτή είναι η program(). Κάθε συνάρτηση που αναπαριστά ένα κανόνα, ελέγχει αν η επόμενη λεκτική μονάδα είναι η προβλεπόμενη από τη γραμματική/και καλεί τις συναρτήσεις που υλοποιούν τους θυγατρικούς κανόνες.

Συναρτήσεις

```
void program();
void programblock();
void block();
void declarations();
void constdecl();
void assignlist();
void assignconst();
void vardecl();
void varlist();
void subprograms();
void procorfunc(enum TPF PF);
void procorfuncbody();
void formalpars();
void formalparlist();
void formalparitem();
void sequence();
void blockorstat();
void statement();
void assignmentstat();
void ifstat(vector<int>& StList);
void elsepart();
void whilestat(vector<int>& StList);
void forstat(vector<int>& StList);
void steppart(string& StList);
void callstat();
void printstat();
void inputstat();
void returnstat();
void selectstat(vector<int>& StList);
void isequalpart(string IId, vector<int> iStList, vector<int> IFalseL);
void actualpars(enum TPF PF, string name);
void actualparlist(enum TPF PF, string name);
void actualparitem();
void condition(vector<int>& TrueL, vector<int>& FalseL);
void boolterm(vector<int>& TrueL, vector<int>& FalseL);
void boolfactor(vector<int>& TrueL, vector<int>& FalseL);
void expression(string& IValueS);
void term(string &IValueT);
void factor(string &IValueF);
```

5. Παραγωγή Ενδιάμεσου Κώδικα

Ο ενδιάμεσος κώδικας είναι μία λίστα με αριθμημένες (με ετικέτες) τετράδες. Βοηθά στην παραγωγή του τελικού κώδικα. Οι τετράδες αναπαριστούν τη γραμμική εκτέλεση του κώδικα του προγράμματος mini_Pascal. Αναπαριστούν την αρχή, το τέλος, τις κλήσεις και τις παραμέτρους των συναρτήσεων ή των διεργασιών. Επίσης αναπαριστούν τις αριθμητικές πράξεις μεταξύ μεταβλητών, τις δηλώσεις τους και τις υλοποιήσεις των statements (πχ for , while , if, select). Κατά την παραγωγή του ενδιάμεσου κώδικα δημιουργούνται προσωρινές μεταβλητές και λίστες με τετράδες αληθείας ή ψεύδους που διευκολύνουν την μετατροπή των statements και πράξεων miniPascal σε σειρά τετράδων.

Τετράδες

“begin_block”,name,_,_	Άνοιγμα block συνάρτησης/διεργασίας name
“end_block”,name,_,_	Κλείσιμο block συνάρτησης/διεργασίας name
“halt”,_,_,_	Τέλος προγράμματος
“jump”,_,_,pos	Μεταφορά της εκτέλεσης στην τετράδα με ετικέτα pos
“:=”,b,_,a	Ανάθεση τιμής a:=b
op,a,b,c	Αριθμητική πράξη c:=a op b με op = { + - * / }
relop,a,b,pos	Σύγκριση a relop b η οποία εάν ισχύει η εκτέλεση μεταφέρεται στην τετράδα με ετικέτα pos
“print”,a,_,_	Εκτύπωση της μεταβλητής a
“input”,a,_,_	Εισαγωγή από το χρήστη και αποθήκευσή της στην a
“ret”,a,_,_	Επιστροφή της μεταβλητής a
“par”,a,tpar,_	Παράμετρος a συνάρτησης/διεργασίας με tpar = RET αν επιστρέφεται, tpar = REF αν εισάγεται με αναφορά, tpar = CV αν εισάγεται με τιμή.
“call”,_,_,name	Κλήση της συνάρτησης name

Μεταβλητές και δομές δεδομένων

```
struct Quad{
        int Qpos;
        string V1;
        string V2;
        string V3;
        string V4;
        Quad(int pos, string i1, string i2, string i3, string i4){
                Qpos=pos;
                V1=i1;
                V2=i2;
                V3=i3;
                V4=i4;
        }
};
```

vector<Quad> Mquads;	Vector αποθήκευσης των τετράδων
Quad *CQuad=NULL;	Current quad
int MQcounter=-1;	Πλήθος τετράδων
int MQnextquad;	Επόμενη τετράδα
string TempVar;	Προσωρινή μεταβλητή
int TempVarCounter=-1;	Βοηθητική μεταβλητή για τη δημιουργία προσωρινών μεταβλητών

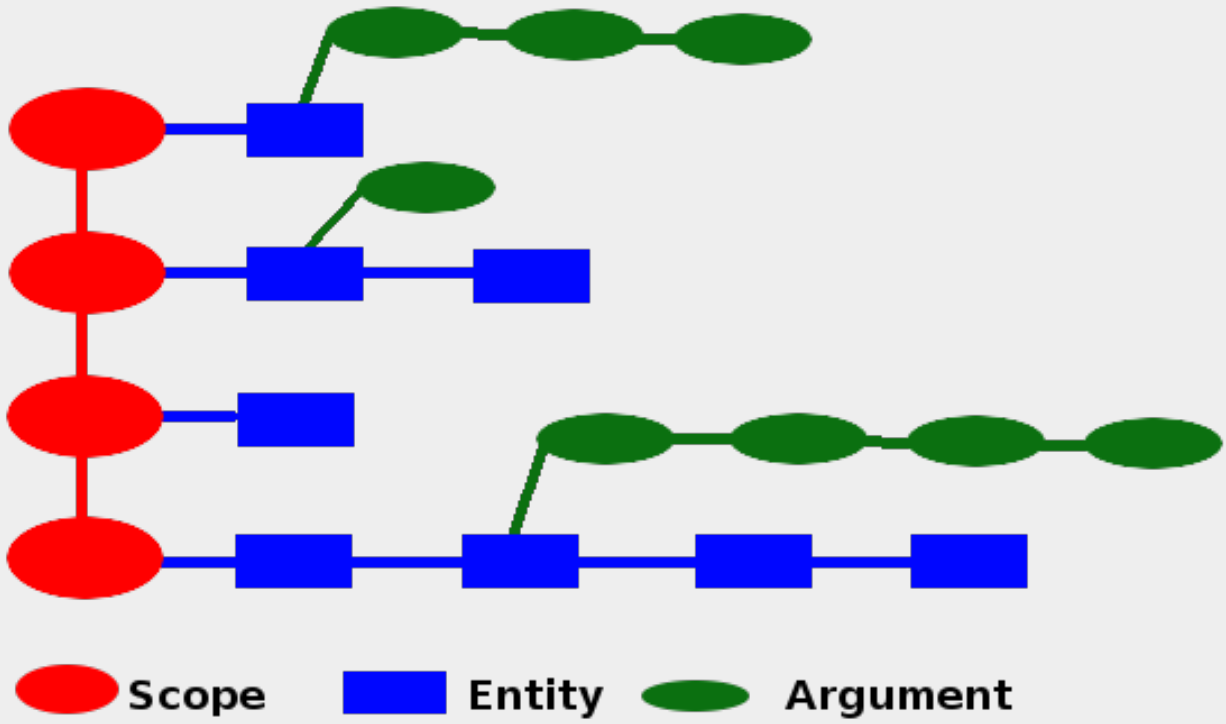
Συναρτήσεις

ο Ενδιάμεσος Κώδικας παράγεται μέσα στις συναρτήσεις του συντακτικού αναλυτή χρησιμοποιώντας τις κάτωθεν συναρτήσεις.

void nextQuad();	Βρίσκει ποιά είναι η ετικέτα της επόμενης τετράδας και τη βάζει στην MQnextquad
void genQuad(string I1, string I2, string I3, string I4);	Δημιουργεί την τετράδα I1, I2, I3, I4 και την προσθέτει στο Mquads
void newTemp();	Δημιουργεί μία νέα προσωρινή μεταβλητή και
την	τοποθετεί στο TempVar.
void merge(vector<int>& v1, vector<int>& v2);	Συνενώνει δύο λίστες αληθείας/ψεύδους
void backpatch(vector<int> BList, int Iqpos);	Ενημερώνει τις τετράδες που ορίζονται στη λίστα αληθείας/ψεύδους BList ώστε να μεταφέρουν την εκτέλεση στην τετράδα με ετικέτα Iqpos

6. Πίνακας Συμβόλων

Ο πίνακας συμβόλων είναι μία δομή η οποία κρατά πληροφορίες και παρουσιάζει σχέσεις μεταξύ του προγράμματος, των συναρτήσεων, των διαδικασιών, των μεταβλητών και των παραμέτρων. Κύριος στόχος του είναι να βοηθήσει τη δημιουργία του εγγραφήματος δραστηριοποίησης. Μπορούμε με βάση αυτό να βρούμε την θέση του κάθε ονόματος (απόσταση από τη main) στη μνήμη. Ο Πίνακας συμβόλων αποτελείται από 3 είδη κόμβων, τα scopes, τα entities και τα arguments.



Μεταβλητές και δομές δεδομένων

```
vector <Scope> MScope;
Scope *CScope=NULL;
Entity *CEntity=NULL;
vector<string> PFnames;      //Program/Funtion names
int FFLength=0;              //function frame length
```

- Scope : Η κεντρική δομή του πίνακα συμβόλων είναι η λίστα των scopes (vector <Scope> MScope).

```
typedef struct Scope Scope;
struct Scope{
        string Name;
        int NestingLevel;
        int CurrentOffset;
        vector <Entity> MEntity;
};
```

Κάθε scope περιέχει ένα όνομα, nesting level (βάθος φωλιάσματος, πόσο απέχει από τη main), current offset και μία λίστα από entities τα οποία περιέχονται στο scope.

- Entity : Κάθε scope εμπεριέχει μία λίστα από entities. Κάθε entity αναπαριστά μία μεταβλητή, μία συνάρτηση, μία διεργασία, μία σταθερά, μία παράμετρος ή μία προσωρινή μεταβλητή.

```
typedef struct Entity Entity;
struct Entity{
        string Name;
        TEnt Etype;
        //variable+constant
        int OffsetVar=0;
        //constant
        string Cvalue;
        //temporary variable
        int OffsetTemp=0;
        //parameter
        int OffsetPar=0;
        TPar Ptype;
        //function
        vector <Argument> MArgument;
        int Fquad;
        int FrLength=0;
};
```

(βοηθητικές δομές δεδομένων)

```
enum TEnt{
        FUNC,           //function
        PAR,            //parameter
        VAR,            //variable
        CVAR,           //constant
        TVAR            //temp variable
};

enum TPF{
        PROC,           //procedure
        FUNCC           //function
};

enum TPar{
        VAL,
        REF,
        DEF
};
enum TPar PDEF=DEF;
```

Κάθε Entity περιέχει τις εξής πληροφορίες.

Name : Το όνομα του entity.

Etype : Τον τύπο του. FUNC για συνάρτηση/διεργασία, PAR για παράμετρο, VAR για μεταβλητή, CVAR για σταθερά, TVAR για προσωρινή μεταβλητή.

OffsetVar : Αν είναι μεταβλητή/σταθερά κρατάμε το offset (απόσταση από την κορυφή της στοίβας).

Cvalue : Αν είναι σταθερά κρατάμε την τιμή της σταθεράς.

OffsetTemp : Αν είναι προσωρινή μεταβλητή κρατάμε το offset (απόσταση από την κορυφή της στοίβας).

OffsetPar , Ptype : Αν είναι παράμετρος κρατάμε το offset (απόσταση από την κορυφή της στοίβας) και τον τρόπο περάσματος της παραμέτρου (με τιμή ή με αναφορά)

Fquad , FrLength , vector <Argument> Margument : Αν είναι συνάρτηση/διεργασία κρατάμε την ετικέτα της πρώτης τετράδας, το μήκος εγγραφήματος δραστηριοποίησης και μία λίστα με Arguments.

- Argument : Αναπαριστά μία παράμετρο συνάρτησης ή διεργασίας.

```
typedef struct Argument Argument;
struct Argument{
        string Name;
        TPar Ptype;
};
```

Κρατείται το όνομα της κάθε παραμέτρου και ο τρόπος περάσματος (με αναφορά ή τιμή)

Συναρτήσεις

Ο πίνακας συμβόλων δημιουργείται μέσα στις συναρτήσεις του Συντακτικού Αναλυτή με τη βοήθεια των κάτωθι συναρτήσεων.

```
void addScope(string IscopeName);
```
Προσθήκη νέου Scope στη λίστα MScope.

```
void addEntity(string IEntityName, enum TEnt ITEnt, string ICValue, int FRLength);
```
Προσθήκη ενός entity στο τρέχον scope.

```
void addArgument(string IArgumentName, enum TPar ITPar);
```
Προσθήκη Argument στο τρέχονς entity.

```
void deleteScope();
```
Διαγραφή ενός scope.

```
void searchVariableS(string name);
```
Αναζήτηση μίας μεταβλητής μέσα στον πίνακα συμβόλων.

```
Entity* searchNameE(string name);
```
Αναζήτηση ενός entity με βάση το όνομα.

```
void startQuadF(string pname);
```
Προσθήκη, στο ανάλογο entity, της ετικέτας της πρώτης τετράδας του κυρίως προγράμματος.

Σημειώσεις

Το εγγράφημα δραστηριοποίησης ενός scope είναι ο χώρος στη μνήμη που αυτό καταλαμβάνει. Εκεί αποθηκεύονται πληροφορίες παραμέτρων, αποτελεσμάτων, συναρτήσεων, διεργασιών και μεταβλητών αυτού του scope. Το framelength είναι το μήκος του εγγραφήματος δραστηριοποίησης σε bytes.

4bytes Διεύθυνση επιστροφής συνάρτησης
4bytes Διεύθυνση στοίβας πατέρα (σύνδεσμος προσπέλασης)
4bytes Διεύθυνση επιστροφής αποτελέσματος
4bytes για κάθε παράμετρο Παράμετροι
4bytes για κάθε μεταβλητή Τοπικές μεταβλητές
4bytes για κάθε μεταβλητή Προσωρινές μεταβλητές

Το offset κάθε entity είναι η απόσταση του entity από την αρχή του scope στο οποίο ανήκει. Το πρώτο entity έχει offset 12 και μετά το κάθε επόμενο entity έχει offset = “offset προηγούμενου entity” + 4

7. Παραγωγή Τελικού κώδικα

Στο στάδιο του τελικού κώδικα παίρνουμε τη λίστα με τις τετράδες που δημιουργήσαμε κατά την παραγωγή του ενδιάμεσου κώδικα, τις μεταφράζουμε σε γλώσσα μηχανής την οποία γράφουμε σε ένα αρχείο Final.obj .

Σημείωση : με τον όρο "γλώσσα μηχανής" αναφερόμαστε όχι στην πραγματική γλώσσα μηχανής (η οποία αποτελείται από 0 και 1) αλλά στη συμβολική γλώσσα (assembly) η οποία θα περιγραφεί παρακάτω.

Γλώσσα Μηχανής

Με τη γλώσσα μηχανής εκτελούμε εντολές χρησιμοποιώντας (αναπαριστώντας) τους καταχωρητές της ALU και κατώτατου επιπέδου εντολές.
Η γλώσσα μηχανής έχει 256 καταχωρητές που τους συμβολίζουμε με R[0], R[1], ... , R[255].
Ο R[0] χρησιμοποιείται ως stack pointer (SP) ο οποίος δείχνει στην αρχή του εγγραφήματος δραστηριοποίησης.
Χρησιμοποιούμε το σύμβολο " $ " ως program counter (PC) .
Η πρόσβαση στη μνήμη επιτυγχάνεται με την έκφραση M[address] . Στη θέση addres μπορούμε να βάλουμε τη διεύθυνση που περιέχεται σε κάποιο καταχωρητή R[x]. πχ M[R[x]], M[R[0]+offset]

Εντολές Γλώσσας Μηχανής

ini tr Εισαγωγή ακεραίου από το χρήστη και αποθήκευσή του στον tr
outi so1 Εκτύπωση περιεχομένων του so1
addi tr,so1,so2 Πρόσθεση so1 + so2 και αποθήκευση του αποτελέσματος στον tr (tr:=so1+so2)
subi tr,so1,so2 Αφαίρεση so1 – so2 και αποθήκευση του αποτελέσματος στον tr (tr:=so1-so2)
muli tr,so1,so2 Πολλαπλασιασμός so1 * so2 και αποθήκευση του αποτελέσματος στον tr (tr:=so1*so2)
divi tr,so1,so2 Διαίρεση so1 * so2 και αποθήκευση του αποτελέσματος στον tr (tr:=so1*so2)

Στις άνωθεν εντολές τα tr, so1, so2 αναπαριστούν καταχωρητές

movi tr,so1 Εκχώριση tr:=so1
Σε αυτή την εντολή τα tr so1 μπορούν να αναπαριστούν τα εξής.

tr	***so1***
μνήμη	*register*
register	*μνήμη*
register	*αριθμός*
register	*R[0] (o SP)*
register	*PC*

jmp addr Άλμα στην addr.
cmpi so1,so2 Εισάγει τους καταχωρητές so1,so2 προς σύγκριση
jb addr Άλμα στην addr εάν so1>so2
jbe addr Άλμα στην addr εάν so1>=so2
ja addr Άλμα στην addr εάν so1<so2
jae addr Άλμα στην addr εάν so1<=so2
je addr Άλμα στην addr εάν so1=so2
jne addr Άλμα στην addr εάν so1<>so2

Στις άνωθεν περιπτώσεις η addr μπορεί να είναι ετικέτα, καταχωρητής, αριθμός ή προσπέλαση μνήμης

Μεταβλητές και δομές δεδομένων

fstream ffinal ("Final.obj")	File stream για εγγραφή στο αρχείο Final.obj
int fLabelf=-1	Σημαία για εισαγωγή (στο αρχείο) του label της πρώτης εκτελέσιμης εντολής
int returnf=-1	Σημαία για τον έλεγχο επιστροφής
int argcounter=0	Μετρητής arguments
Quad *FCQuad=NULL	Τρέχουσα τετράδα τελικού κώδικα (δείχνει σε μία τετράδα της MQuads)
int enestingLevel=0	Κρατάει το nesting level που ψάχνουμε (ενημερώνεται από τη searchNameE())

Συναρτήσεις

bool isNum(string& isstr) — Ελέγχει εάν ένα string περιέχει έναν αριθμό (κάθε char είναι νούμερο)

void gnlvcode(string gtname) — Μεταφέρει στον R[255] τη διεύθυνση μίας μη τοπικής μεταβλητή

```
ffinal<<"\t movi\t R[255],\t M[4+R[0]]\n";                  movi R[255],M[4+R[0]]
for(int i=enestingLevel;i<CScope->NestingLevel;i++)
    ffinal<<"\t movi\t R[255],\t M[R[255]+4]\n";            movi R[255],M[4+R[255]]
ffinal<<"\t movi\t R[254],\t "<<goffset<<"\n";              movi R[254],offset
ffinal<<"\t addi\t R[255],\t R[255],\t R[254]\n";           addi R[255],R[255],R[254]
```

void loadvr(string var,int reg) — Φόρτωση αριθμού/μεταβλητής var σε έναν καταχωρητή R[reg]

- Αν var είναι αριθμός
  movi R[reg],var
- Αν var είναι global μεταβλητή (μεταβλητή του κυρίως προγράμματος)
  movi R[reg],M[600+offset]
- Αν var είναι τοπική μεταβλητή ή παράμετρος που περνάει με τιμή και nesting level ίσο με το τρέχον ή προσωρινή μεταβλητή
  movi R[reg],M[offset+R[0]]
- Αν var είναι παράμετρος που περνάει με αναφορά και nesting level ίσο με το τρέχον
  movi R[255],M[offset+R[0]]
  movi R[reg],M[R[255]]
- Αν var είναι τοπική μεταβλητή ή παράμετρος που περνάει με τιμή και nesting level μικρότερο από το τρέχον
  gnlvcode(var)
  movi R[reg],M[R[255]]
- Αν var είναι παράμετρος που περνάει με αναφορά και nesting level μικρότερο από το τρέχον
  gnlvcode(var)
  movi R[255],M[R[255]]
  movi R[reg],M[R[255]]

void storerv(string var,int reg) — Αποθήκευση μιάς μεταβλητής στη μνήμη

- Αν var είναι global μεταβλητή (μεταβλητή του κυρίως προγράμματος)
  movi M[600+offset],R[reg]
- Αν var είναι τοπική μεταβλητή ή παράμετρος που περνάει με τιμή και nesting level ίσο με το τρέχον ή προσωρινή μεταβλητή
  movi M[offset+R[0]],R[reg]

- Αν var είναι παράμετρος που περνάει με αναφορά και nesting level ίσο με το τρέχον
  movi M[offset+R[0]],R[255]
  movi M[R[255]],R[reg]
- Αν var είναι τοπική μεταβλητή ή παράμετρος που περνάει με τιμή και nesting level μικρότερο από το τρέχον
  gnlvcode(var)
  movi M[R[255]]R[reg]
- Αν var είναι παράμετρος που περνάει με αναφορά και nesting level μικρότερο από το τρέχον
  gnlvcode(var)
  movi M[R[255]],R[255]
  movi M[R[255]],R[reg]

void generateFinalCode()	Η κύρια συνάρτηση που παράγει τον τελικό κώδικα. Ελέγχει συνεχώς (μέχρι το τέλος των τετράδων) τις τετράδες και ανα περίπωση παράγει τον τελικό κώδικα σε γλώσσα μηχανής
void createFFirstLabel()	Τοποθετεί το label της πρώτης εκτελέσιμης εντολής
void checkFunCall(string funcname)	Ελέγχει την κλήση της συνάρτησης και τη σχέση της με την συνάρτηση που την καλεί (εάν αυτή υπάρχει)

8. Χρήση Μεταφραστή

Ο Μεταφραστής γίνεται compile με την εντολή.

```
g++ -std=c++11  ./Compiler_3final_Vlachogkountis_Euaggelopoulos.cpp -o ./Compiler
```

Εκτελείται με μία μοναδική παράμετρο η οποία είναι το αρχείο (test) προς μετάφραση, το αρχείο που περιέχει τον κώδικα mini-Pascal.

```
./Compiler  test
```

Στο τέλος της εκτέλεσης θα δημιουργηθεί ένα αρχείο Final.obj που περιέχει το πρόγραμμα μεταφρασμένο σε γλώσσα μηχανής.