

## Η Γλώσσα Προγραμματισμού Cimple

Η Cimple είναι μια μικρή, εκπαιδευτική, γλώσσα προγραμματισμού. Θυμίζει τη γλώσσα C, από την οποία και αντλεί ιδέες και δομές, αλλά είναι αρκετά πιο μικρή, τόσο στις υποστηριζόμενες δομές, όσο φυσικά και σε προγραμματιστικές δυνατότητες.

Το όνομα Cimple προέρχεται από τη γλώσσα C, τη λέξη *implementation*, ενώ ηχεί όπως η λέξη *simple*. ώστε να τονιστεί η απλότητά της.

Παρόλο που οι προγραμματιστικές της δυνατότητες είναι μικρές, η εκπαιδευτική αυτή γλώσσα περιέχει πλούσια στοιχεία και η κατασκευή του μεταγλωττιστή της έχει να παρουσιάσει αρκετό ενδιαφέρον, αφού περιέχονται σε αυτήν πολλές εντολές που χρησιμοποιούνται από δημοφιλείς γλώσσες, καθώς και κάποιες πρωτότυπες. Η Cimple υποστηρίζει δομές, όπως η *while* και η *if-else*, αλλά και τις πρωτότυπες και ενδιαφέρουσες στην υλοποίηση *forcase* και *incase*. Επίσης, υποστηρίζει συναρτήσεις και διαδικασίες, μετάδοση παραμέτρων με αναφορά και τιμή, αναδρομικές κλήσεις. Επιτρέπει φώλιασμα στη δήλωση συναρτήσεων κάτι που λίγες γλώσσες υποστηρίζουν (το υποστηρίζει η Pascal, δεν το υποστηρίζει η C).

Από την άλλη πλευρά, η Cimple δεν υποστηρίζει βασικά προγραμματιστικά εργαλεία, όπως η δομή *for* ή τύπους δεδομένων όπως οι πραγματικοί αριθμοί και οι συμβολοσειρές ή οι πίνακες. Οι παραλήψεις αυτές έχουν γίνει για εκπαιδευτικούς λόγους, ώστε να απλουστευτεί η διαδικασία κατασκευής του μεταγλωττιστή, μία απλοποίηση, όμως, που έχει να κάνει μόνο με τη μείωση των γραμμών κώδικα και όχι με τη δυσκολία κατασκευής του ή την εκπαιδευτική αξία της ανάπτυξης.

Έτσι, θα κατασκευάσουμε και θα παρακολουθήσουμε βήμα-βήμα την ανάπτυξη ενός μεταγλωττιστή Cimple, χωρίς τη χρήση οποιονδήποτε εργαλείων ανάπτυξης, αλλά με τη χρήση μόνο μίας γλώσσας προγραμματισμού. Ο μεταγλωττιστής θα είναι πλήρως λειτουργικός και θα παράγει, ως τελική γλώσσα, τη γλώσσα *assembly* του επεξεργαστή MIPS. Χρησιμοποιώντας έναν εξομοιωτή του επεξεργαστή MIPS θα είναι δυνατόν να τρέξουμε την *assembly* που θα δημιουργεί ο μεταγλωττιστής. Η επιλογή της *assembly* του MIPS ως γλώσσα στόχο, έχει να κάνει με εκπαι-

δευτικούς λόγους, αφού η επιλογή ενός εμπορικού επεξεργαστή δεν έχει να προσθέσει κάτι εκπαιδευτικά.

Τα αρχεία της Cimple έχουν κατάληξη .ci

## 0.1 Λεκτικές Μονάδες

Το αλφάβητο της Cimple αποτελείται από:

- τα μικρά και κεφαλαία γράμματα της λατινικής αλφαβήτου (A,...,Z και a,...,z),
- τα αριθμητικά ψηφία (0,...,9),
- τα σύμβολα των αριθμητικών πράξεων (+, -, \*, /),
- τους τελεστές συσχέτισης (<, >, =, <=, >=, <>)
- το σύμβολο ανάθεσης (:=)
- τους διαχωριστές (;, “”, :)
- τα σύμβολα ομαδοποίησης ([, ], (, ) , { , })
- του τερματισμού του προγράμματος (.)
- και διαχωρισμού σχολίων (#)

Τα σύμβολα [, ] χρησιμοποιούνται στις λογικές παραστάσεις όπως τα σύμβολα (, ) στις αριθμητικές παραστάσεις.

Οι δεσμευμένες λέξεις είναι:

program	declare				
if	else	while			
switchcase	forcase	incase	case	default	
not	and	or			
function	procedure	call	return	in	inout
input	print				

Οι λέξεις αυτές δεν μπορούν να χρησιμοποιηθούν ως μεταβλητές. Οι σταθερές της γλώσσας είναι ακέραιες σταθερές που αποτελούνται από προαιρετικό πρόσημο και από μία ακολουθία αριθμητικών ψηφίων. Οι ακέραιες σταθερές πρέπει να έχουν τιμές από  $-(2^{32} - 1)$  έως  $2^{32} - 1$ .

Τα αναγνωριστικά της γλώσσας είναι συμβολοσειρές που αποτελούνται από γράμματα και ψηφία, αρχίζοντας όμως από γράμμα. Κάθε αναγνωριστικό αποτελείται από τριάντα το πολύ γράμματα. Αναγνωριστικά με περισσότερους από 30 χαρακτήρες θεωρούνται λανθασμένα.

Οι λευκοί χαρακτήρες (tab, space, return) αγνοούνται και μπορούν να χρησιμοποιηθούν με οποιονδήποτε τρόπο χωρίς να επηρεάζεται η λειτουργία του μεταγλωττιστή, αρκεί βέβαια, να μην βρίσκονται μέσα σε δεσμευμένες λέξεις, αναγνωριστικά, σταθερές.

Το ίδιο ισχύει και για τα σχόλια, τα οποία πρέπει να βρίσκονται ανάμεσα στα σύμβολα #.

## 0.2 Μορφή προγράμματος

Κάθε πρόγραμμα ξεκινάει με τη λέξη κλειδί `program`. Στη συνέχεια ακολουθεί ένα αναγνωριστικό (όνομα) για το πρόγραμμα αυτό και τα τρία βασικά μπλοκ του προγράμματος: οι δηλώσεις μεταβλητών (declarations), οι συναρτήσεις και διαδικασίες (subprograms), οι οποίες μπορούν και να είναι φωλιασμένες μεταξύ τους, και οι εντολές του κυρίως προγράμματος (statements). Η δομή ενός προγράμματος Cimple φαίνεται παρακάτω. Προσέξτε την τελεία στο τέλος.

```
program id
    declarations
    subprograms
    statements
.
```

## 0.3 Τύποι και δηλώσεις μεταβλητών

Ο μοναδικός τύπος δεδομένων που υποστηρίζει η Cimple είναι οι ακέραιοι αριθμοί. Η δήλωση γίνεται με την εντολή `declare`. Ακολουθούν τα ονόματα των αναγνωριστικών χωρίς καμία άλλη δήλωση, αφού γνωρίζουμε ότι πρόκειται για ακέραιες μεταβλητές και χωρίς να είναι αναγκαίο να βρίσκονται στην ίδια γραμμή. Οι μεταβλητές χωρίζονται μεταξύ τους με κόμματα. Το τέλος της δήλωσης αναγνωρίζεται με το ελληνικό ερωτηματικό. Επιτρέπεται να έχουμε περισσότερες των μία συνεχόμενες χρήσεις της `declare`.

## 0.4 Τελεστές και εκφράσεις

Η προτεραιότητα των τελεστών από τη μεγαλύτερη στη μικρότερη είναι:

- Πολλαπλασιαστικοί: `*`, `/`
- Προσθετικοί: `+`, `-`
- Σχεσιακοί: `=`, `<`, `>`, `<>`, `<=`, `>=`
- Λογικοί: `not`
- Λογική σύζευξη: `and`
- Λογική διάζευξη: `or`

## 0.5 Δομές της γλώσσας

### 0.5.1 Εκχώρηση

ID := expression

Χρησιμοποιείται για την ανάθεση της τιμής μιας μεταβλητής ή μιας σταθεράς, ή μιας έκφρασης σε μία μεταβλητή.

### 0.5.2 Απόφαση if

```
if (condition)
    statements1
[ else
    statements2 ]
```

Η εντολή απόφασης `if` εκτιμά εάν ισχύει η συνθήκη *condition* και εάν πράγματι ισχύει, τότε εκτελούνται οι εντολές *statements<sub>1</sub>* που το ακολουθούν. Το `else` δεν αποτελεί υποχρεωτικό τμήμα της εντολής και γι' αυτό βρίσκεται σε αγκύλη. Οι εντολές *statements<sub>2</sub>* που ακολουθούν το `else` εκτελούνται εάν η συνθήκη *condition* δεν ισχύει.

### 0.5.3 Επανάληψη while

```
while (condition)
    statements
```

Η εντολή επανάληψης `while` επαναλαμβάνει τις εντολές *statements*, όσο η συνθήκη *condition* ισχύει. Αν την πρώτη φορά που θα αποτιμηθεί η *condition* το αποτέλεσμα της αποτίμησης είναι ψευδές, τότε οι *statements* δεν εκτελούνται ποτέ.

### 0.5.4 Επιλογή switchcase

```
switchcase
    (case (condition) statements1 ) *
    default statements2
```

Η δομή `switchcase` ελέγχει τις *condition* που βρίσκονται μετά τα `case`. Μόλις μία από αυτές βρεθεί αληθής, τότε εκτελούνται οι αντίστοιχες *statements<sub>1</sub>* (που ακολουθούν το *condition*). Μετά ο έλεγχος μεταβαίνει έξω από την `switchcase`. Αν, κατά το πέρασμα, καμία από τις `case` δεν ισχύσει, τότε ο έλεγχος μεταβαίνει στην `default` και εκτελούνται οι *statements<sub>2</sub>*. Στη συνέχεια ο έλεγχος μεταβαίνει έξω από την `switchcase`.

### 0.5.5 Επανάληψη `forcase`

```
forcase
    (case (condition) statements1 ) *
    default statements2
```

Η δομή επανάληψης `forcase` ελέγχει τις *condition* που βρίσκονται μετά τα `case`. Μόλις μία από αυτές βρεθεί αληθής, τότε εκτελούνται οι αντίστοιχες *statements<sub>1</sub>* (που ακολουθούν το *condition*). Μετά ο έλεγχος μεταβαίνει στην αρχή της `forcase`. Αν καμία από τις `case` δεν ισχύει, τότε ο έλεγχος μεταβαίνει στη `default` και εκτελούνται οι *statements<sub>2</sub>*. Στη συνέχεια ο έλεγχος μεταβαίνει έξω από την `forcase`.

### 0.5.6 Επανάληψη `incase`

```
incase
    (case (condition) statements1 ) *
```

Η δομή επανάληψης `incase` ελέγχει τις *condition* που βρίσκονται μετά τα `case`, εξετάζοντας τις κατά σειρά. Για κάθε μία για τις οποίες η αντίστοιχη *condition* ισχύει εκτελούνται οι *statements* που ακολουθούν το *condition*. Θα εξεταστούν με τη σειρά όλες οι *condition* και θα εκτελεστούν όλες οι *statements* των οποίων οι *condition* ισχύουν. Αφότου εξεταστούν όλες οι `case`, ο έλεγχος μεταβαίνει έξω από τη δομή `incase`, εάν καμία από τις *statements* δεν έχει εκτελεστεί, ή μεταβαίνει στην αρχή της `incase`, εάν έστω και μία από τις *statements* έχει εκτελεστεί.

### 0.5.7 Επιστροφή τιμής συνάρτησης

```
return (expression)
```

Χρησιμοποιείται μέσα σε συναρτήσεις για να επιστραφεί το αποτέλεσμα της συνάρτησης, το οποίο είναι το αποτέλεσμα της αποτίμησης του *expression*.

### 0.5.8 Έξοδος δεδομένων

```
print (expression)
```

Εμφανίζει στην οθόνη το αποτέλεσμα της αποτίμησης του *expression*.

### 0.5.9 Είσοδος δεδομένων

```
input (ID)
```

Ζητάει από τον χρήστη να δώσει μία τιμή μέσα από το πληκτρολόγιο. Η τιμή που θα δώσει θα μεταφερθεί στην μεταβλητή *ID*.

### 0.5.10 Κλήση διαδικασίας

```
call functionName(actualParameters)
```

Καλεί μία διαδικασία.

### 0.5.11 Συναρτήσεις και διαδικασίες

Η Cimple υποστηρίζει συναρτήσεις και διαδικασίες. Για τις συναρτήσεις η σύνταξη είναι:

```
function ID (formalPars)
{
    declarations
    subprograms
    statements
}
```

ενώ για τις διαδικασίες:

```
procedure ID (formalPars)
{
    declarations
    subprograms
    statements
}
```

Η `formalPars` είναι η λίστα των τυπικών παραμέτρων. Οι συναρτήσεις και οι διαδικασίες μπορούν να φωλιάσουν η μία μέσα στην άλλη. Οι κανόνες εμβέλειας ακολουθούν τους κανόνες της PASCAL. Η επιστροφή της τιμής μιας συνάρτησης γίνεται με την `return`.

Η κλήση μιας συνάρτησης, γίνεται μέσα από τις αριθμητικές παραστάσεις σαν τελούμενο. π.χ.

```
D = a + f(in x)
```

όπου  $f$  η συνάρτηση και  $x$  παράμετρος που περνάει με τιμή.

Η κλήση μιας διαδικασίας γίνεται με την `call`. π.χ.

```
call f(inout x)
```

όπου  $f$  η διαδικασία και  $x$  παράμετρος που περνάει με αναφορά.

## 0.6 Μετάδοση παραμέτρων

Η Cimple υποστηρίζει δύο τρόπους μετάδοσης παραμέτρων:

- *με τιμή*: Δηλώνεται με τη λεκτική μονάδα `in`. Αλλαγές στην τιμή της δεν επιστρέφονται στο πρόγραμμα που κάλεσε τη συνάρτηση
- *με αναφορά*: Δηλώνεται με τη λεκτική μονάδα `inout`. Κάθε αλλαγή στη τιμή της μεταφέρεται αμέσως στο πρόγραμμα που κάλεσε τη συνάρτηση

Στην κλήση μίας συνάρτησης οι πραγματικοί παράμετροι συντάσσονται μετά από τις λέξεις κλειδιά `in` και `inout`, ανάλογα με το αν περνούν με τιμή ή αναφορά.

## 0.7 Κανόνες εμβέλειας

*Καθολικές* ονομάζονται οι μεταβλητές που δηλώνονται στο κυρίως πρόγραμμα και είναι προσβάσιμες σε όλους. *Τοπικές* είναι οι μεταβλητές που δηλώνονται σε μία συνάρτηση ή διαδικασία και είναι προσβάσιμες μόνο μέσα από τη συγκεκριμένη συνάρτηση ή διαδικασία. Κάθε συνάρτηση ή διαδικασία, εκτός των τοπικών μεταβλητών, των παραμέτρων της και των καθολικών μεταβλητών, έχει επίσης πρόσβαση και στις μεταβλητές που έχουν δηλωθεί σε συναρτήσεις ή διαδικασίες προγόνους ή και σαν παραμέτρους αυτών.

Ισχύει ο δημοφιλής κανόνας ότι, αν δύο (ή περισσότερες) μεταβλητές ή παράμετροι έχουν το ίδιο όνομα και έχουν δηλωθεί σε διαφορετικό επίπεδο φωλιάσματος, τότε οι τοπικές μεταβλητές και παράμετροι υπερκαλύπτουν τις μεταβλητές και παραμέτρους των προγόνων, οι οποίες με τη σειρά τους υπερκαλύπτουν τις καθολικές μεταβλητές.

Μία συνάρτηση ή διαδικασία έχει δικαίωμα να καλέσει τον εαυτό της και όποια συνάρτηση βρίσκεται στο ίδιο επίπεδο φωλιάσματος με αυτήν, της οποίας η δήλωση προηγείται στον κώδικα.

## 0.8 Η γραμματική της Cimple

Ακολουθεί η γραμματική της Cimple η οποία δίνει και την ακριβή περιγραφή της γλώσσας:

```
# "program" is the starting symbol
program      :    program ID block .

# a block with declarations, subprogram and statements
block        :    declarations subprograms statements

# declaration of variables, zero or more "declare" allowed
declarations :    ( declare varlist ; ) *

# a list of variables following the declaration keyword
varlist      :    ID ( , ID ) *
               |    ε
```

```

# zero or more subprograms allowed
subprograms      :      ( subprogram )*

# a subprogram is a function or a procedure,
# followed by parameters and block
subprogram       :      function ID ( formalparlist ) block
                    |      procedure ID ( formalparlist ) block

# list of formal parameters
formalparlist    :      formalparitem ( , formalparitem )*
                    |      ε

# a formal parameter ("in": by value, "inout" by reference)
formalparitem    :      in ID
                    |      inout ID

# one or more statements
statements       :      statement ;
                    |      { statement ( ; statement )* }

# one statement
statement        :      assignStat
                    |      ifStat
                    |      whileStat
                    |      switchcaseStat
                    |      forcaseStat
                    |      incaseStat
                    |      callStat
                    |      returnStat
                    |      inputStat
                    |      printStat
                    |      ε

# assignment statement
assignStat       :      ID := expression

# if statement
ifStat           :      if ( condition ) statements elsepart

elsepart         :      else statements
                    |      ε

```



```

# while statement
whileStat      :   while ( condition ) statements

# switch statement
switchcaseStat:   switchcase
                  ( case ( condition ) statements )*
                  default statements

# forcase statement
forcaseStat    :   forcase
                  ( case ( condition ) statements )*
                  default statements

# incase statement
incaseStat     :   incase
                  ( case ( condition ) statements )*

# return statement
returnStat     :   return( expression )

# call statement
callStat       :   call ID( actualparlist )

# print statement
printStats     :   print( expression )

# input statement
inputStat      :   input( ID )

# list of actual parameters
actualparlist  :   actualparitem ( , actualparitem )*
                  |
                   $\epsilon$ 

# an actual parameter ("in": by value, "inout" by reference)
actualparitem  :   in expression
                  |
                  inout ID

# boolean expression
condition      :   boolterm ( or boolterm )*

```

```

# term in boolean expression
boolterm      :    boolfactor ( and boolfactor )*

# factor in boolean expression
boolfactor    :    not [ condition ]
                |    [ condition ]
                |    expression REL_OP expression

# arithmetic expression
expression    :    optionalSign term ( ADD_OP term )*

# term in arithmetic expression
term          :    factor  ( MUL_OP factor )*

# factor in arithmetic expression
factor        :    INTEGER
                |    ( expression )
                |    ID idtail

# follows a function or procedure (parethnesis and parameters)
idtail        :    ( actualparlist )
                |    ε

# sumbols "+" and "-" (are optional)
optionalSign  :    ADD_OP
                |    ε

# lexer rules: relational, arithentic operations, integers and ids
REL_OP        :    = | <= | >= | > | < | <>
                ;
ADD_OP        :    + | -
MUL_OP        :    * | /
INTEGER       :    [0-9]+
ID            :    [a-zA-Z][a-zA-Z0-9]*

```

## 0.9 Παραδείγματα κώδικα

Παρακάτω δίνονται τρία μικρά παραδείγματα σε γλώσσα Cimple: ο υπολογισμός του παραγοντικού, η ακολουθία Fibonacci και η μέτρηση των ψηφίων ενός αριθμού.

Υπολογισμός παραγοντικού:

```
program factorial

# declarations #
declare x;
declare i,fact;

# main #
{
    input(x);
    fact:=1;
    i:=1;
    while (i<=x)
    {
        fact:=fact*i;
        i:=i+1;
    };
    print(fact);
}.
```

Η ακολουθία Fibonacci:

```
program fibonacci

declare x;

function fibonacci(in x)
{
    return (fibonacci(in x-1)+fibonacci(in x-2));
}

# main #
{
    input(x);
    print(fibonacci(in x));
}.
```

Μέτρηση των ψηφίων ενός αριθμού:

```
program countDigits

  declare x, count;

  # main #
  {
    input(x);
    count := 0;
    while (x>0)
    {
      x := x/10;
      count := count+1;
    };
    print(count);
  }.
```