Η Γλώσσα Προγραμματισμού *C-imple*

Γεώργιος Μανής
Πανεπιστήμιο Ιωαννίνων
Πολυτεχνική Σχολή
Τμήμα Μηχανικών Η/Υ και Πληροφορικής

Φεβρουάριος 2022

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

Παρόλο που οι προγραμματιστικές της δυνατότητες είναι μικρές, η εκπαιδευτική αυτή γλώσσα προγραμματισμού 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 προγραμματισμού, καθώς και κάποιες πρωτότυπες. Η *C-imple* υποστηρίζει δομές όπως οι δημοφιλείς `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 και διαδικασιών.

Από την άλλη όμως πλευρά, η *C-imple* δεν προσφέρει 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. Είναι μία απλοποίηση που έχει να κάνει μόνο με τη μείωση των γραμμών κώδικα που πρέπει να γραφεί και όχι με τη δυσκολία της κατασκευής ή την εκπαιδευτική αξία της διαδικασίας ανάπτυξης.

Στο κεφάλαιο αυτό θα οριστεί η γλώσσα *C-imple*. Θα παρουσιαστεί η δομή ενός προγράμματος σε γλώσσα *C-imple*, οι λεκτικές μονάδες που συνθέτουν ένα πρόγραμμα, οι λέξεις κλειδιά που είναι δεσμευμένες, οι τελεστές που είναι διαθέσιμες, οι προγραμματιστικές δομές που υποστηρίζονται, οι κανόνες φωλιάσματος συναρτήσεων και διαδικασιών, οι κανόνες εμβέλειας και ο τρόπος περάσματος παραμέτρων.

Θα κατασκευάσουμε και θα παρακολουθήσουμε βήμα-βήμα την ανάπτυξη ενός μεταγλωττιστή *C-imple*, χωρίς τη χρήση οποιονδήποτε εργαλείων ανάπτυξης, αλλά μόνο μίας γλώσσας προγραμματισμού. Ο μεταγλωττιστής θα είναι πλήρως λειτουργικός και θα παράγει, ως τελική γλώσσα, τη γλώσσα assembly του επεξεργαστή RISC-V. Χρησιμοποιώντας έναν εξομοιωτή του επεξεργαστή RISC-V θα είναι δυνατόν να τρέξουμε την assembly που θα δημιουργεί ο μεταγλωττιστής. Η επιλογή της assembly του RISC-V ως γλώσσα στόχο, έχει να κάνει με εκπαιδευτικούς λόγους, αφού η επιλογή ενός εμπορικού επεξεργαστή δεν έχει να προσθέσει κάτι εκπαιδευτικά.

Τα αρχεία της *C-imple* έχουν κατάληξη `.ci`

1.1 Λεκτικές Μονάδες

Το αλφάβητο της *C-imple* 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. Ο μεταγλωττιστής λαμβάνει υπόψη του μόνο τα τριάντα πρώτα 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 μέσα σε σύμβολα `#`

1.2 Μορφή προγράμματος

Κάθε πρόγραμμα ξεκινάει με τη λέξη κλειδί `program`. Στη συνέχεια ακολουθεί ένα αναγνωριστικό (όνομα) για το πρόγραμμα αυτό. Μέσα σε άγκιστρα τοποθετούνται τα τρία βασικά μπλοκ του προγράμματος: οι δηλώσεις των 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 *C-imple* φαίνεται παρακάτω:

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

1.3 Τύποι και δηλώσεις μεταβλητών

Ο μοναδικός τύπος δεδομένων που υποστηρίζει η *C-imple* 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`.

1.4 Τελεστές και εκφράσεις

Η προτεραιότητα των τελεστών από τη μεγαλύτερη στη μικρότερη είναι:

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

1.5 Δομές της γλώσσας

1.5.1 Εκχώρηση

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

Σύνταξη:

```
ID := expression
```

1.5.2 Απόφαση if

Η εντολή απόφασης `if` εκτιμά εάν ισχύει η συνθήκη *condition* και εάν πράγματι ισχύει, τότε εκτελούνται οι εντολές $statements_1$ που το ακολουθούν. Το `else` δεν αποτελεί υποχρεωτικό τμήμα της εντολής και γι' αυτό βρίσκεται σε αγκύλη. Οι εντολές $statements_2$ που ακολουθούν το `else` εκτελούνται εάν η συνθήκη *condition* δεν ισχύει.

Σύνταξη:

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

1.5.3 Επανάληψη while

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.

Σύνταξη:

```
while (condition)
        statements
```

1.5.4 Επιλογή switchcase

Η δομή `switchcase` ελέγχει τις *condition* που βρίσκονται μετά τα `case`. Μόλις μία από αυτές βρεθεί αληθής, τότε εκτελούνται οι αντίστοιχες $statements_1$ (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 $statements_2$. Στη συνέχεια ο έλεγχος μεταβαίνει έξω από την `switchcase`.

Σύνταξη:

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

1.5.5 Επανάληψη forcase

Η δομή επανάληψης `forcase` ελέγχει τις *condition* που βρίσκονται μετά τα `case`. Μόλις μία από αυτές βρεθεί αληθής, τότε εκτελούνται οι αντίστοιχες $statements_1$ (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 $statements_2$. Στη συνέχεια ο έλεγχος μεταβαίνει έξω από την `forcase`.

Σύνταξη:

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

1.5.6 Επανάληψη incase

Η δομή επανάληψης `incase` ελέγχει τις *condition* που βρίσκονται μετά τα `case`, εξετάζοντας τες κατά σειρά. Για κάθε μία από αυτές που η αντίστοιχη *condition* ισχύει, εκτελούνται οι αντίστοιχες *statements* (που ακολουθούν το *condition*). Θα εξεταστούν 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`, έαν έστω και μία από τις *statements* έχει εκτελεστεί.

Σύνταξη:

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

1.5.7 Επιστροφή τιμής συνάρτησης

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

Σύνταξη:

```
return (expression)
```

1.5.8 Έξοδος δεδομένων

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

Σύνταξη:

```
print (expression)
```

1.5.9 Είσοδος δεδομένων

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

Σύνταξη:

```
input (ID)
```

1.5.10 Κλήση διαδικασίας

Καλεί μία διαδικασία.

Σύνταξη:

```
call functionName(actualParameters)
```

1.5.11 Συναρτήσεις και διαδικασίες

Η *C-imple* 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 και οι κανόνες εμβέλειας είναι όπως της PASCAL. Η επιστροφή της τιμής μιας συνάρτησης γίνεται με την `return`.

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

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

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

1.6 Μετάδοση παραμέτρων

Η *C-imple* υποστηρίζει δύο τρόπους μετάδοσης παραμέτρων:

- *με σταθερή 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.

1.7 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.

1.8 Η γραμματική της *C-imple*

Ακολουθεί η γραμματική της *C-imple* η οποία δίνει και την ακριβή περιγραφή της γλώσσας:

```
# "program" is the starting symbol
# followed by its name and a block
# Every program ends with a fullstop
program         →   program ID
                    block
                    .

# a block consists of declarations, subprograms and statements
block           →   {
```

```
                    declarations
                    subprograms
                    blockstatements
                }


# declaration of variables
# Kleene star implies zero or more "declare" statements
declarations  →   ( declare varlist ; )*

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

# zero or more subprograms
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
# one or more parameters are allowed
formalparlist →    formalparitem
                   ( , formalparitem )*
              |    ε

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

# one or more statements
# more than one statements should be grouped with brackets
statements    →    statement ;
              |    {
                        statement
                        ( ; statement )*
                   }

# statements considered as block (used in program and subprogram)
blockstatements→   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

# else part is optional
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
printStat     →    print( expression )

# input statement
inputStat     →    input( ID )
```

```
# list of actual parameters
actualparlist →    actualparitem
                   ( , actualparitem )*
               |    ε

# 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
# describes parethneses and parameters
idtail        →    ( actualparlist )
               |    ε

# symbols "+" and "-" (are optional)
optionalSign  →    ADD_OP
               |   ε

##########################

# lexer rules: relational, arithentic operations,
#              integer values and ids

REL_OP        →    = | <= | >=  | > | < | <>
```

```
ADD_OP          →     + | -

MUL_OP          →     * | /

INTEGER         →     [0-9]+

ID              →     [a-zA-Z][a-zA-Z0-9]*
```

1.9 Παραδείγματα κώδικα

Παρακάτω δίνονται τρία μικρά παραδείγματα σε γλώσσα *C-imple*: 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);
}.
```

Υπολογισμός του αθροίσματος $1 + 2 + \ldots + N$*, με την forcase*:

```
program summation
{
    declare x,sum;

    # main #
    input(x);
    sum :=0;

    forcase
        case(x>0)
        {
            sum := sum+x;
            x := x-1
        }
        default
            print(sum);
}.
```

Υπολογισμός των 30 πρώτων πρώτων αριθμών:

```
program primes
{
    # declarations for main #
    declare i;

    function isPrime(in x)
    {
        # declarations for isPrime #
        declare i;

        function divides(in x, in y)
        {
            # body of divides #
            if (y = (y/x)*x)
                return (1);
            else
                return (0);
        }

        # body of isPrime #
        i:=2;
        while (i<x)
        {
            if (divides(in i, in x)=1)
                return(0);;
            i := i + 1
        };
        return(1)
    }

    # body of main #
    i := 2;
    while (i<=30)
        if (isPrime(in i)=1)
            print(i);;
}.
```

Παρατηρήστε ότι σε δύο σημεία του κώδικα η *C-imple* απαίτησε από τον προγραμματιστή να τοποθετήσει δύο συνεχόμενα ελληνικά ερωτηματικά (μετά από το δεύτερο `return` και μετά από το `print`). Ο κανόνας στη *C-imple* λέει ότι κάθε εκτελούμενη εντολή τερματίζει με ερωτηματικό. Δεν είναι απαραίτητο το ερωτηματικό όταν η εντολή αυτή είναι η τελευταία ενός μπλοκ. Στην περίπτωση της `return` στο σημείο αυτό τερματίζουν οι εντολές `return`, `if` και `while`. Η `while` ως τελευταία στο μπλοκ δεν απαιτεί ερωτηματικό (αν βάλουμε δεν είναι συντακτικό λάθος). Άρα χρειαζόμαστε τουλάχιστον δύο ερωτηματικά εκεί. Στην περίπτωση της `print` στο σημείο αυτό τερματίζουν οι εντολές `print`, `if` και `while`. Η `while` ως τελευταία στο μπλοκ δεν απαιτεί ερωτηματικό. Άρα και εδώ χρειαζόμαστε τουλάχιστον δύο ερωτηματικά.