Programming Language Design
Department of Computer Science
Dr. Francisco Ortin
francisco.ortin@mtu.ie

Specification of the C-- language

The language to be designed and implemented by the student must provide some mandatory features. Some other optional features are proposed to obtain a higher mark. The language follows a C-based syntax.

Minimum Set of Features

The following features are mandatory to pass the module. They are worth 70% of the mark.

A program is defined as a sequence of variable and function definitions. They can be intermixed. The syntax of a variable definition is a type followed by an identifier and one final ";" character.

```
int a; // example variable definition

double realNumber; // another variable definition
```

Functions are defined by specifying the return type, the function identifier, and a list of comma-separated parameters between `(` and `)`. The return type and parameter types must be built-in (i.e., no arrays). The function body goes between `{` and `}`. The bodies of functions are sequences of local variable definitions, followed by sequences of statements. Both must end with the ";" character.

```
int add(int a, int b) {

    int temp;

    temp = a + b;

    return temp;

}
```

The last and mandatory function is the "main" function, which returns `void` and receives no parameters.

Built-in types are `int`, `double` and `char`. Array types can be created with the `[]` type constructor, following the Java syntax but specifying the size of the array with an integer literal (like C).

```
int[10] v; // v is an array of 10 ints

double[5][10] w; // w is a 2-dimensional array of doubles
```

The `write` statement writes data in the standard output (console). Its syntax is the `write` keyword followed by an expression. The read statement is similar, but using the `read` keyword.

```
read v[i];

write w[i][a+b];
```

Assignments are built with two expressions separated by the = operator.

The `if/else` and `while` statements follow the C syntax (recall that `else` block is optional). In C--, no local variables can be defined inside `if/else` and `while` blocks.

The `return` statement is also supported (the expression after the `return` keyword is mandatory; i.e., `return;` is not a valid statement in this language).

A function invocation may be an expression or a statement. A procedure (a function returning `void`) invocation is always a statement.

```
void main() {
        int result;
        result = add(1, 2); // invocation as an expression
        add(result, 1); // invocation as a statement
}
```

The cast expression follows the C syntax (casts only allows built-in types).

Expressions can be built with the following structure:

- Integer, real and character literals without sign.
- Identifiers.
- The following operators, applied to one or two expressions (descending order of precedence):

`( )`	Non associative
`[]`	Non associative
- (unary)	Non associative
`!`	Non associative
`* / %`	Left associative
`+ -`	Left associative
`> >= < <= != ==`	Left associative
`&& ||`	Left associative

Additional Optional Features

The 30% of the reminding evaluation could be obtained by adding the implementation of the following features to the language. Please, notice that these optional features must be implemented **in order** (i.e., feature *n* should not be implemented if feature *n-1* has not been implemented yet). Additionally, if you implement one feature for the mid-term exam, that does not force you to implement it for the final exam.

Multiple definition of variables (5% additional evaluation)

Variables with the same type could be declared in the same variable definition.

```
int a, b, c;
double[10] realVector, anotherOne;
```

Multiple write and read statements (5% additional evaluation)

Write and read statements could specify a list of (at least one) expressions.

```
write a+b, v[45+c];
read realNumber, w[6][t];
```

Structs (20% additional evaluation)

The record (struct) type is provided by the language. Struct variables could be local and global. No typedef construction is provided. Fields in structs can be obtained with the "`.`" operator.

```
struct {
      int day, month;
      int year;
} date;
void main() {
      struct { char first; int second; } pair;
      date.year = 2018;
      date.month = 8;
      date.day = w[1][date.month];
      pair.second = date.day;
}
```

Structs could be nested: a field of a record could be another record. They can also be combined with any other type, including arrays.

```
void main() {
        struct {
                int age;
                struct { int day, month; int year; } dateOfBirth;
                char[256] name, surname;
        } [20] students; // 'students' is an array of 20 structs

        students[0].dateOfBirth.day = 10;
}
```