

|  |  |
| --- | --- |
|  | *User manual* |
|  |  |
|  | Flores, L;Salazar, Y.  Compiler design  5/7/18 |

Index

[Lexis and syntax 2](#_Toc513425102)

[Example programs 4](#_Toc513425103)

[1. Iterative Fibonacci series 4](#_Toc513425104)

[2. Iterative factorial function 5](#_Toc513425105)

[3. Recursive factorial function 5](#_Toc513425106)

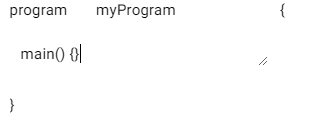
[4. Array find 6](#_Toc513425107)

# Lexis and syntax

MobProl is a language aimed at young audiences who would like to start learning about programming with simple programs with basic concepts or want to try programming ideas on the go.

On the lexis, MobProl supports four different types for variables and functions: *int* (Integer numbers)*, decim* (Decimal values)*, bool* (Boolean operators)*, void* (Aimed to be used in functions). You will find that, aside from the name “decim” for our floating-point values, most other reserved words in the syntax are congruent with most popular programming languages. This language is meant to serve as a starting point for a programmer.

The syntax of a common program requires to write a main function as well as the name of the program. Thus, the simplest program one could write is as follows:



MobProl’s syntax supports the following:

* Primitive, array and matrix variables declaration
  + var int myInt;
  + var bool myBoolArr[10];
  + var decim myDecimMat[10][2];
* Variable assignation
  + myInt = 5;
  + myDecimMat[8][2] = 9.5;
* Functions and recursive calls with pass by value parameters
  + func int myFunc(int myParam) {}
  + myVar = myFunc(myParamToSend);
* Prints
  + print (myVar);
  + print (1);
  + print (“Hello World”);
* Read for user input
  + read (myVar);
* Expressions
  + myInt = myOtherInt + 4;
* Conditionals
  + if (myCondition) {}
  + if (myCondition) {} else {}
* Cycles
  + while (myCounter <= myArrLength) {}

# Example programs

MobProl is aimed to write simple programs with simple user interaction. Following, some example programs that one could write:

## Iterative Fibonacci series

program Fibonacci {

main() {

var int n;

var int t1;

var int t2;

var int nextTerm;

var int i;

t1 = 0;

t2 = 1;

nextTerm = 0;

print("Terms to print for Fibonacci");

read (n);

i = 0;

while (i <= n) {

if (i == 1) {

print (t1);

} else {

if (i == 2) {

print (t2);

}

else {

nextTerm = t1 + t2;

t1 = t2;

t2 = nextTerm;

print (nextTerm);

}}

i++;

}

}  
}

## Iterative factorial function

program FactorialIterative {

func int factorial(int num) {

var int result;

result = 1;

while (num > 0) {

result = result \* num;

num--;

}

return result;

}

main() {

var int factStart;

print("Pick a number to get its factorial");

read(factStart);

print(factorial(factStart));

}  
}

## Recursive factorial function

program FactorialRecursive {

func int factorial(int num) {

if (num > 1) {

return (num \* factorial(num-1));

}

else {

return num;

}

}

main() {

var int factStart;

print("Pick a number to get its factorial");

read(factStart);

print(factorial(factStart));

}  
}

## Array find

program ArrayFind {

main() {

var int a[10];

var int aLength;

var int i;

var int numToSearch;

var int numIndex;

aLength = 10;

print("Please input 10 numbers for your array");

i = 1;

while (i <= aLength) {

read(a[i]);

i++;

}

i = 1;

while (i <= aLength) {

print(a[i]);

i++;

}

print("Please tell me the number you would like to find");

read(numToSearch);

numIndex = 0;

i = 1;

while (i <= aLength && numIndex < 1) {

if (a[i] == numToSearch) {

numIndex = i;

}

i++;

}

if (numIndex > 0) {

print("Found your number on index");

print(numIndex);

}

else {

print("Sorry I could not find your number on the array provided");

}

}  
}