LANGUAGE REFERENCE MANUAL

TEAM - 10

CS18B037 Swetha R, CS18B043 Shruti Priya, CS18B044 Sai Datta, CS18B045 Pranathi W

1. Program constructs

Keywords:

- *int*, *float*, *if*, *else*, *while*, *function*, *print*, *input*, *break*, *continue*, *return*

Identifiers:

- Can contain upper and lower case alphabets, digits and underscores. Spaces or special symbols like !, @, #, etc… are not to be part of an identifier.
- Must start with a letter
- Can be of any length
- Are case sensitive
- Cannot be the same as any of the keywords
- Examples :
    - Allowed: Count, int_count, count1
    - Not allowed: count@1, count 1, 1count, _count

Datatypes:

- Integer (int)
- Float (float)
- String (variables of this type cannot be declared but, string literals can be passed as arguments to the print function)

Arrays:

- Static multidimensional arrays - no restriction on the number of dimensions but each dimension must be an integer constant (no dynamic allocation)

Expressions:

- Operators
    - Arithmetic: addition, subtraction, multiplication, division
    - Relational: greater than, greater than or equal, less than, less than or equal to, equal to, not equal to,
    - Logical: and, or
- The precedence order of operators:
  (*, /) > (+, -) > (>=, <=, >, <) > (&&) > (||)

- Expressions can be made with any combination of these operators. (complex expressions)

Comments:

- Single line comments, written starting with //

```
// The comment text
```

Statements:

Every statement must end with a semicolon (;)

- Declaration
    - Starting with the type, any number of variables or arrays of the type can be declared while separated by commas.
    - Arrays as mentioned are static, hence the dimensions need to be specified using integer literals(constants).
    - Variables can be declared and initialized to a constant/ expression, the same is not the case with array declarations.
    - Arrays can only be initialised entry by entry using appropriate indices.
    - Examples:
        - Allowed:

        - *int* i, j = 0 , k = j;
        - *int* array[2][3][5];
    - Not allowed:
        - int i, float j;
        - int array[n];
- Assignment
    - Variable/array element to be assigned on the left, the value/ expression/ input function/ any user-defined function call on the right separated by the equal to (=) sign.
- Conditional (If - else)
    - Conventional if-else statement like in C (elseif not included)
    - No restrictions on the number of levels of nesting
    - Every else must have a corresponding if
    - The statements inside the if and else must be enclosed within a block, starting and ending with flowery brackets {...} even if there is only 1 statement.

- Loop (While)
    - Conventional while statement like in C
    - Break and continue statements can be written inside.
    - No restrictions on the number of levels of nesting
    - The statements inside the while loop must be enclosed in a block, starting and ending with flowery brackets {...} even if there is only 1 statement.
- Print
    - Can be used as

---

print(arg_1, arg_2,..., arg_n);

    - Each arg can be the identifier corresponding to a variable, or an array element or a string literal and is separated by a comma.

- Input
  - Can be used as

---

var = *input*(type);

  - The type can be any of int or float.
  - Can only be used on the RHS of an assignment statement.
- Return
  - Can be used as

---

return var; (or) *return*;

  - Used to return from a function
- Break
  - Can be used as

---

break;

  - Breaks from the while loop it is found in.
  - Is ignored when used outside while loops

- Continue
  - Can be used as

---

continue;

  - Continues to the next iteration of the while loop it is found in.
  - Is ignored if found outside while loops

Functions:

- A function definition must follow this syntax:

---

function return_type function_name (type$_1$ var1, type$_2$ var2, … , type$_n$ varn)

{

Stmts…

}

  - The function declaration must start with the keyword 'function', followed by a return value type, then by the function identifier and arguments enclosed in parentheses ()
  - If the function does not return a value, the identifier (function name) must immediately follow the keyword 'function'.
  - The function name cannot be the same as a keyword or an already globally declared identifier/function.
  - Functions can have any number of arguments, each specified with their type and identifier, separated by commas before the function block starts
  - Arrays are not valid function arguments in function definitions.
  - The function block contains the function code enclosed within the flowery brackets {…}.
  - No function call is allowed inside the function block
  - Function has access to variables that are declared globally (before the function definition) along with the ones declared in the function.
- A function call must follow this syntax:

```
function_name(var1, var2, …, varn);    //or
var = function_name(var1, var2, …, varn);
```

- Existing variables, constants or array elements (NOT entire arrays) can be arguments in a function call
- Array as a whole cannot be an argument, but a specific entry can be passed.
- The changes made to function arguments in the function are not reflected in the actual arguments passed in the function call (pass by value only)
- Tip: If a function is required to work on an array, Since passing an array as an argument is not allowed, this array can be declared globally before the function definition and accessed within the function.
- Examples:
  - Intializearray();
  - N = Fibonacci(i);

Block structured/nested blocks:

- Scope of variables declared in a specific block is limited to that block and sub-blocks of the block.
- A block is created as part of function definitions, if-else statements or while statements. Simple blocks cannot be created.
- The whole program is considered to be a global block (much like in Python) in which the mentioned blocks can be created (refer to the sample programs)

2. Sample programs

Program 1:

```
function int fibo(int n) {
        int first = 0, second = 1, i = 2;
        if(n==1)
        { return 0; }
        while(i < n) {
                int third = first + second;
                first = second;
                second = third;
                i = i+1;
        }
        return second;
}
print("enter number ");
int n = input(int);
int ans = fibo(n);
print("\n", n, "th fibonacci: ", ans);
// end of fibonacci code
print("\n enter an int");
int f = input(int), result = 0;
if(f < 0 ) {
        result = 2 * (f <= -1 && f >= -3);
        if(result == 2) { print("\nYay!"); }
        else {print("\nNot yay"); }
}
else {
        result = -2 * (f || 1);
        if(result == -2) { print("\nHurray"); }
}
```

Program 2:

```
int a[3][4];
int i=0, j=0;
// initialising the array
while(i<3)
{
    j=0;
    while(j<4)
    {
        a[i][j] = i*j;
        j=j+1;
    }
   i = i+1;
}
i=0;
j=0;
// printing the array
while(i<3)
{
    j=0;
    print("\n i= " , i);
    while j<4)
    {
        print("\n     j= ",j,"   a[i][j]: ", a[i][j]);
        j=j+1;
    }
    i = i+1;
}
```