Language Manual

Note: The numbers in brackets next to the heading of a section indicates which phase it is supposed to be implemented in.

Identifiers (1)

- Identifiers can be **variable names** or **function names**, and are defined and used by users.
- An accepted identifier is a combination of the following characters:
  - [a-z], [A-Z], [0-9], "_"
  - And all the identifiers should start with a letter i.e [a-zA-Z]
- Examples:

```
abc - accepted
Name - accepted
sum2 - accepted
_max - not accepted
1var - not accepted
dollar$ - not accepted
```

Keywords (1)

- Keywords are reserved by the language for special purposes and cannot be used as an identifier.
- `int, char, float, void, and, or, not, if, else, elif, while, for, switch, case, break, return, continue.`
- Using any of these words in the wrong context will give a compile error.

Data types

- User can declare variable of the following types:
  - **Primitive types:** int (1), char (6), float (7),
  - **Derived types (6):** array, string
- Default values:
  - `int -> 0`
  - `char -> '\0'`
  - `float -> 0.0`
  - `array -> same as its primitive type`
  - `string -> ""`

Note: Any int/float other than 0 and any char with ascii value other than 0 will be treated as true, otherwise false.

Operations

- **Arithmetic operators (1, 3):**
  - Addition (+), Subtraction (-), Multiplication (*), Division (/)
  - For division, the divisor should be non-zero.
  - The operands for all the arithmetic operations are expected to be int/float. But in case they are char, implicit type casting will be done. If implicit type casting is not possible, an appropriate error will be displayed.

- **Bitwise operations (2):**
  - & (and) , | (or) , ^ (xor) , << (left shift) , >> (right shift) , ~ (negation)
  - Bitwise operators can be used on any primitive data type to manipulate the bit representation of the value.
- **Comparison operators (4):**
  - `>, <, >=, <=, ==, !=`
  - Comparison operators can be used on any primitive data type
- **Logical operators (3):**
  - AND (&&), OR (||), NOT (!)

Operator Precedence

Operator	Associativity
Unary -, ~	left
*, /, %	left
+, -	left
<<, >>	left
<=, >=, <, >	left
==, !=	left
&	left
^	left
|	left

&&	left
||	left
=	right

Conditional Statements (4)

- The `if, else, elif` statements.
- An `elif` statement will be executed only if all the preceding `if` and `elif` statements are evaluated as false. Similarly for the `else`.
- Usage:

```
if(<condition>){
}
elif(<condition>){
}
elif(<condition>){
}
.
.
else{
}
```

Switch-case (10)

- Usage:

```
switch(<variable>){
      case(<value1>){
            //case 1
      }
      case(<value2>){
            //case 2
      }
      default{
            //default case
      }
}
```

- `<variable>` and `<value>` should be of the same type.
- Users are advised to use a `break` statement at the end of a case block so as to not let the control go to the next case block after executing the current block.
- In case the `break` statement is not present at the end of a case block, the control reaches the next block and executes it as well until it reaches a break statement or all the cases are exhausted.
- The default case is executed only when none of the above cases are matched with the given variable.
- Example:

```
char cc = 'b';
switch(cc){
    case('a'){
        output(cc, char);
        break;
    }
    case('b'){
        output(cc, char);
        break;
    }
    default{
        output(cc, char);
        break;
    }
}
```

Iterative statements (5)

- Used to repeat a routine until a condition is evaluated as false.
- **For-loop:**
  - Usage:

```
for(<init>; <condition>; <inc/dec>){
    // for loop body
}
```

  - `<init>` denotes the initialization of a variable that will be used in the `<condition>`.
  - `<condition>` is used to check if the loop should end or not.
  - `<inc/dec>` statement is used to write logic to increment or decrement the value of the loop variable initialized before.
  - Example:

```
for(int a=1; a<=10; a=a+1){
    output(a, int);
}
```

- **While-loop:**
  - Usage:

```
while(<condition>){
    // while loop body
}
```

  - Example:

```
int a = 0;
while(a<10){
    output(a, int);
    a = a+1;
}
```

- **Note**: In iterative statements, `break` statements can be used to exit from the loop at any point, and `continue` statements can be used to skip the rest of the code in the loop to proceed to the next iteration.

IO statements (2)

- Users can read and write to **stdout** and **stdin**.
- Usage:

```
input(<variable_name>, <type>);
output(<variable_name>/<literal>/<expr>, <type>);
```

- Examples:

```
input(num, int);
output(x, char);
```

Declaration statements

- Possible with and without initialization.
- Can declare multiple variables of the same type in one declaration statement.
- For primitive types (1, 6, 7):

```
<type>  <identifier>  /,//without  initialization(except strings)
<type> <identifier> = <expression>; //with initialization
```

- For arrays (6):

```
<type> <identifier>[<size>];
<type> <identifier>[<size>] = {<values separated by comma>};
```

- Examples:

```
int a;
```

```
char my_char = ‘w’;
int arr[10];
int arr[5] = {5,2,3,4,1};
```

Assignment statements

- Users can assign values to primitive type variables and array elements. The LHS must have only one variable.
- Usage:

```
<identifier> = <expression>           (1)
<identifier>[<index>] = <expression>  (8)
```

- Examples:

```
x = x+y
a = (a-b)/2*c
arr[3] = a+b/2
```

Functions (10)

- User needs to define the parameters, their types and return type when defining a function.
- Cannot pass arrays or strings to function calls.
- Usage:

```
<return_type> <function_name> (<param1_type> <param1_name>,
<param2_type> <param2_name>, ….) {
      // function body
}
```

- Example:

```
int sum(int a, int b){ //calculates the sum of two numbers
      return a+b;
}
```

Main function (1)

- Mandatory for all programs. Main function doesn’t take any parameters and can return anything (doesn’t matter).

```
void main(){
      // body
}
```

Type Casting (9)

- **Implicit type casting:**
  1. Implicit type casting **widens** the type of a variable or a constant.
  2. Data type priority: **char < int < float**
  3. Examples:

```
type(5 + 3.5) -> float
type('a' - 1) -> int
type(1 + 'a') -> int
```

- **Explicit type casting:**
  1. When widening, this is the same as impliciting type casting. But may result in data loss if the type is narrowed.
  2. The decimal part is truncated if casted from **float to int**.
  3. When converted from **char to int**, the ASCII value of the character is returned.
  4. Usage:

```
(<type>) <variable/constant>
```

  5. Examples:

```
int num = (int)'a';
char cc = (char)97;
```

Comments (1)

- **Single-line comments** start with "**//**", and anything written after that in the line is considered as part of the comment.
- **Multi-line comments** start with "**/***" and end with "***/**", and any thing written between them is considered as part of the comment.
- Examples:

```
// this is a single-line comment

/*   this is
     a multi-line
     comment
*/
```

Example programs

1. Fibonacci series

```
void main() {
```

```
        //initialize the array with zeros
        int fibonacci[6] = {0,0,0,0,0,0};
        fibonacci[1] = 1;
        int i = 2;
        for(int i=2; i<6; i=i+1){
             fibonacci[i] = fibonacci[i-1] + fibonacci[i-2];
             output(fibonacci[i], int);
        }
    }
```

2. Sort all the number in an array in ascending order

```
    void main(){

        int i, j, a, n, number[30];

        output("Enter the value of N \n", string);
        input(n, int);

        output("Enter the numbers  \n", string);

        for (i = 0; i < n; i=i+1)
            input(number[i], int);

        for (i = 0; i < n; i=i+1) {
            for (j = i + 1; j < n; j=j+1){
             if (number[i] > number[j]) {
                 a =  number[i];
                 number[i] = number[j];
                 number[j] = a;
             }
            }
        }

        output("The numbers arranged in ascending order are
    given below \n", string);

        for (i = 0; i < n; i=i+1)
            output(number[i], int);
```

```
}
```

3. Print the sum of all the elements in an array

```
void main(){

    int arr[100], size, i, sum = 0;

    output("enter array size \n", string);
    input(size, int);

    output("enter array elements \n", string);
    for(i = 0; i < size; i=i+1)
       input(arr[i], int);

    for(i = 0; i < size; i=i+1)
          sum = sum + arr[i];


    output("sum of the array = ", string);
    output(sum, int);

}
```

4. Print the largest number in the array

```
 void main(){

     int arr[5] = {10, 324, 45, 90, 98};
     int n = 5;

     int max = arr[0];

     for (int i = 1; i < n; i=i+1)
         if (arr[i] > max)
```

```
                    max = arr[i];

                output("largest in given array is ", string);
                output(largest(arr,n), int);

            }
```

5. Print “yes” if the number is greater than 100, otherwise “no”

```
        void main(){

            int i;
            output("enter the value of i\n", string);
            input(i, int);

            if(i>100)
                output("yes", string);
            else
                output("no", string);

        }
```

6. Print the all numbers between two numbers given as input

```
        void main(){

            int a,b;
            output("enter two numbers \n",string);

            input(a, int);
            input(b, int);

            while(a<b-1){
                a=a+1;
                output(a, int);
            }
        }
```

Grammar Rules

```
Program         :   func_list

func_list       :   func_list func | ;

func            :   func_prefix OF stmt_list CF

func_prefix     :   data_type ID OC param_list CC

param_list      :   param_list COMMA param | param | ;

param           :   data_type ID | data_type ID OS CS COLON INT ID;

stmt_list       :   stmt stmt_list | ;

stmt            :   declaration | assign SCOL | expr SCOL |
return_stmt SCOL | if_stmt | while_loop_stmt | for_loop_stmt | BREAK
SCOL | CONTINUE SCOL | switch_stmt;

declaration     :   data_type ID SCOL | data_type ID ASSIGN expr SCOL
;

return_stmt     :   RETURN expr ;

data_type       :   INT | FLOAT | CHAR ;

/* Expressions */
expr            :   expr ADD expr | expr SUBTRACT expr | expr MULTIPLY
expr | expr DIVIDE expr | expr LE expr | expr GE expr | expr LT expr |
expr GT expr | expr EQ expr | expr NE expr | expr AND expr | expr OR
expr | expr MODULO expr | expr BITAND expr | expr BITOR expr | expr
XOR expr | unary_expr | primary_expr | postfix_expr ;

postfix_expr    :   func_call ;

unary_expr      :   unary_op primary_expr ;
```

```
primary_expr    :   ID | const | OC expr CC ;

unary_op        :   ADD | SUBTRACT | NOT | NEGATION ;

const           :   INT_NUM | CHARACTER ;

assign          :   ID ASSIGN expr ;

/* if-elif-else */
if_stmt         :   IF OC expr CC OF stmt_list CF elif_stmt  else_stmt
;

elif_stmt       :   ELIF OC expr CC OF stmt_list CF elif_stmt | ;

else_stmt       :   ELSE OF stmt_list CF | ;

/* Switch */
switch_stmt     :   SWITCH OC ID CC OF case_stmt_list default_stmt CF
;

case_stmt_list  :   case_stmt case_stmt_list | ;

case_stmt       :   CASE OC const CC COLON stmt_list ;

default_stmt    :   DEFAULT COLON stmt_list | ;

/* While */
while_loop_stmt :   WHILE OC expr CC OF stmt_list CF ;

/* For */
for_loop_stmt   :   FOR OC assign SCOL expr SCOL ;

/* Function call */
func_call       :   ID OC arg_list CC ;

arg_list        :   arg COMMA arg_list | arg | ;

arg             :   expr;
```

Plan - Phase wise

Phase 1	Main function, Type (int), Simple assignment, Arithmetic (+, -)
Phase 2	Bitwise (&, |, ^, <<, >>, ~) and IO statements
Phase 3	Arithmetic (*, /), Logical (AND, OR, NOT)
Phase 4	Conditional (if, if-else, if-elif-else), Comparison operators
Phase 5	Iterative (for, while)
Phase 6	Type (char), Type (Arrays), Type(Strings)
Phase 7	Type (float)
Phase 8	Array Assignment
Phase 9	Explicit, Implicit Type Casting
Phase 10	Functions, Switch-Case (conditional)

Sample Programs

Simple

```
// Various operations using switch-case
int main(){

    int a;
    int b;
    int c;

    output("Enter \\\"a\\\" value = ");
    input(a);

    output("Enter \\\"b\\\" value = ");
    input(b);

    output("Select the operation: \n1. Sum (+)\n2. Product (*)\n3. Bitwise And (&)\n");
```

```
	int op;
	input(op);

	output("Result: ");

	switch(op){
	case(1):
	c=a+b;
	break;
	case(2):
	c=a*b;
	break;
	case(3):
	c=a&b;
	break;
	default:
	output("Invalid\n");
	}
	output(c);
	return 0;
}


// Ascii values
int main(){
	char arr[5] = {'v', 'w', 'x', 'y', 'z'};
	int i;
	for(i=0; i<5; i=i+1){
		output("Ascii value of ");
		output(arr[i]);
		output(" is ");
		int temp = arr[i];
		output(temp);
		output(".\n");
	}

	string str = "This is a string.\n";

	output(str);
	return 0;
}


// Check whether it's a prime number
```

```
int main(){

        int number;

        output("Enter the number: ");
        input(number);

    int i;
    for (i = 2; i <= number / 2; i=i+1) {
        if (number % i != 0){
            continue;
        }
        else{
            output("It is a prime number.\n");
                                return 0;
                        }
    }
        output("It is not a prime number.\n");

    return 0;
}

int main(){

        float f;
        output("Enter a float value: ");
        input(f);

        int t = f;
        float x = t;
        float y = f-x;

        output("Floor:");
        output(x);

        output("\n");
        if(y>0){
        x=x+1;
        }
        output("Ceil:");
        output(x);

        return 0;
}
```

Moderate

```
// 3x3 matrix multiplication
int main(){

	int a[9];
	int b[9];

	output("Enter the contents of matrix 1:\n");

	int i;

	for(i=0; i<9; i=i+1){
		int t;
		input(t);
		a[i] = t;
	}

	output("Enter the contents of matrix 2:\n");

	for(i=0; i<9; i=i+1){
		int t;
		input(t);
		b[i] = t;
	}

	int c[9];

	int row;
	for(row=0; row<3; row=row+1){
		int col;
		for(col=0; col<3; col=col+1){
			int sum = 0;
			int k;
			for(k=0; k<3; k=k+1){
				sum = sum + a[row*3+k]*b[k*3+col];
			}
			c[row*3+col] = sum;
```

```
                }
        }

        for(i=0; i<9; i=i+1){
                output(c[i]);
                if((i+1)%3 == 0){
                        output("\n");
                }else{
                        output(" ");
                }
        }
        return 0;
}


// Queue - push, pop
int main(){

        int q[100];
        int qs = 0;

        string msg = "\nOperations:\n1. Push back to the Queue\n2. Pop from the front of
the Queue\n3. End\n\n";

        while(1){

                output("Queue: ");
                int i;
                for(i=0; i<qs; i=i+1){
                        output(q[i]);
                        output(" " );
                }
                output("\n");

                output(msg);

                int op;
                input(op);

                switch(op){
                        case(1):
                                output("Enter the number: ");
                                int t;
                                input(t);
```

```
                                        q[qs] = t;
                                        qs = qs+1;
                                case(2):
                                        for(i=1; i<qs; i=i+1){
                                                q[i-1] = q[i];
                                        }
                                        qs = qs-1;
                                case(3):
                                        break;
                        }

                }
                return 0;
}
```

Complex

```
// Combinations using recursion

void main(){

        output("Enter n: ");
        int n;
        input(n);

        output("Enter r: ");
        int r;
        input(r);

        int res = nCr(n, r);

        output("nCr: ");
        output(res);
        output("\n");
}

int nCr(int n, int r){

        if(n < r){
                return 0;
        }
        elif(n == r){
```

```
		return 1;
	}
	else{
		if(r==0){
			return 1;
		}
		return nCr(n-1, r-1) + nCr(n-1, r);
	}
}


// BST - Insert only
int main(){

	int tree[63]; // 6 levels
	int i;
	for(i=0; i<100; i=i+1){
		tree[i] = 0;
	}

	while(1){

		// Print the tree
		output("Tree: \n\n");

		int l = 0;
		int n = 1;
		int m = 0;

		for(l=0; l<6; l=l+1){
			int i;
			for(i=0; i<n; i=i+1){
				output(tree[m+i]);
				output(" ");
			}
			output("\n");
			m = m + n;
			n = n * 2;
		}

		output("\n\n");


		// -----------------
```

```
                output("Insert: ");
                int in;
                input(in);

                int cur = 0;
                while(tree[cur] != 0){
                        if(in < tree[cur]){
                                cur = 2*cur + 1;
                        }else{
                                cur = 2*cur + 2;
                        }
                }

                tree[cur] = in;
        }
}

// Selection Sort

void main(){

        int arr[10];

        output("Enter array elements: ");
        int i;
        for(i=0; i<10; i=i+1){
                input(arr[i]);
                // arr[i] = 9-i;
        }

        for(i=0; i<9; i=i+1){
                int min_i = i;
                int j;
                for(j=i+1; j<10; j=j+1){
                        if(arr[j] < arr[min_i])
                                min_i = j;
                }
                int temp = arr[min_i];
                arr[min_i] = arr[i];
                arr[i] = temp;
        }

        output("Sorted array: ");
        for(i=0; i<10; i=i+1){
```

```
                output(arr[i]);
                output(" ");
        }
        output("\n");

}
```