HIGH LEVEL PROGRAMMING LANGUAGE

TABLE OF CONTENTS

Language Specification

Syntax and Semantics

The structure of *our Programming language* programs follows a consistent syntax. Statements are terminated by semicolons (;), and blocks of code are enclosed in curly braces (**{...}**) .

Programming Paradigm

- Procedural Programming paradigm

Variables and Data Types

Variables are declared using the keywords. *Our programming language* supports following data types -

int : Represents integers.
bool : Represents boolean variables.
char : Represents character

Arrays

Arrays are collections of values of the same data type. They can be declared as follows -

```
var myArr = int[5]; // Declare an array of 5 integers - var has to be
replaced with int,char bool
```

Control Structures

If statement

```
if (condition) {
    // code to execute if condition is true
}
else {
    // code to execute if condition is false
}
```

while Loop

```
while (condition)
{
    // Loop body
}
```

for Loop

```
for (var i = 0; i < 10; i++)
{
    // Loop body
}
```

Functions

Functions are defined based on the return type -

```
int add(int a, int b)
{
    return a + b;
}
```

Keywords and Operators

Keywords

- if, else, while, for, return, switch, case, break, continue
- int, bool, char
- this, mac, main, Float, String

Operators

Arithmetic	+, -, *, /
Comparison	==, !=, <, <=, >, >=
Logical	&&, ||, =
Bitwise	&,|

Grammar Rules

1. They define the syntax of a programming language
2. Grammar rules play a crucial role in parsing, which is the process of analyzing the source code and converting it into a structured representation that can be further processed by the compiler.
3. It's typically defined using formal notation, such as Backus-Naur Form (BNF) or Extended Backus-Naur Form (EBNF). These notations consist of symbols, non-terminals, terminals, and production rules.

Grammar Rules List

program → declList | '#include' ‘<’ library ‘>’ declList

class → ‘class’ ID ‘{‘ classVarDecl classCons classFuncList ‘}’

classVarDecl → varDecl | null

classCons → ID (parms) “{“ consStatementList “return this” “}”

consStatementList → consStatementList consStatement | null

consStatement → “this” “.” exp “=” exp “;”

classFuncList → classFunclList , funcDecl | funcDecl

library → ID ‘.h’ | ID

declList → declList decl | decl

decl → varDecl | funDecl

varDecl → dataType varDeclList ;

varDeclList → varDeclList , varDeclInit | varDeclInit

varDeclInit → varDeclId | varDeclId ‘=’ simpleExp

varDeclId → ID | ID [NUMCONST]

dataType → int | bool | char

funDecl → dataType ID (parms) stmtList | “void” ID (parms) stmtList

parms → parmList | NULL

parmList → parmList ‘,’ parmTypeList | parmTypeList

parmTypeList → dataType parmId

parmId → ID | ID [ ]

stmt → expStmt | compoundStmt | selectStmt | iterStmt | returnStmt | breakStmt | macStmt

expStmt → exp ; | ;

selectStmt → 'if' '(' simpleExp ')' stmt | 'if' '(' simpleExp ')' stmt 'else' stmt | ‘switch’ ‘(‘ simpleExp ‘)’ stmt iterStmt → ‘while’ '(' simpleExp ')' stmt | ‘for’ (exp ‘;’ simpleExp ‘;’ exp) stmt

labeledStmt → ‘case’ constant ‘:’ stmt | ‘default’ ‘:’ stmt

compoundStmt → '{' localDecls stmtList '}' | localDecls stmtList

macStmt → mac ID ',' ID ',' ID ';'

localDecls → localDecls VarDecl | NULL

stmtLit → stmtList stmt | NULL

returnStmt → return ; | return exp ;

breakStmt → break ;

exp → mutable = exp | mutable += exp | mutable −= exp | mutable ∗= exp | mutable/= exp | mutable ++ | mutable −− | exp + exp | exp - exp | exp * exp | simpleExp

simpleExp → simpleExp or andExp | andExp

andExp → andExp and unaryRelExp | unaryRelExp

unaryRelExp → not unaryRelExp | relExp

relExp → minmaxExp relop minmaxExp | minmaxExp

relop → <= | < | > | >= | == | ! =

minmaxExp → minmaxExp minmaxop sumExp | sumExp

minmaxop → > | <

sumExp → sumExp sumop mulExp | mulExp

sumop → + | −

mulExp → mulExp mulop unaryExp | unaryExp

mulop → ∗ | / | %

unaryExp → unaryop unaryExp | factor

unaryop → − | ∗ | ? | ~

factor → immutable | mutable

mutable → ID | ID [exp]

immutable → (exp) | call | constant

call → ID (args)

args → argList | NULL

argList → argList , exp | exp

constant → NUMCONST | CHARCONST | true | false

Errors

Syntax Errors -

- int x = 6 // error: missing ;
- Int x = 6; // error: Int is not a type

Semantic Errors -

- a+b=c; // semantic error

Run-time Errors -

- n/num // error - Division by zero, if num==0 at any point of time during execution

Libraries

How to define → #include <library_name>

Array

```
class Array {
    /** Constructs a new Array of the given size. */
    Array (int size) {
        return Memory.alloc(size);
    }
    /** De-allocates the array and frees its space. */
     void dispose() {
         Memory.deAlloc(this);
        return;
    }
}
```

I/O

#include <ProgInOut>

```
class ProgInOut
{
      void progin();
      void progout();
}
```

Math

#include <Math>

```
Class Math{
    int division(int num1, int num2) {
        if (num1 == 0)
            return 0;
        if (num2 == 0)
            return INT_MAX;
        bool negResult = false;
        if (num1 < 0) {
            num1 = -num1 ;
            if (num2 < 0)
                num2 = - num2 ;
            else
                negResult = true;
        }
        else if (num2 < 0)  {
            num2 = - num2 ;
            negResult = true;
        }

        int quotient = 0;
        while (num1 >= num2)  {
            num1 = num1 - num2 ;
            quotient ;
        }
        if (negResult)
            quotient = - quotient ;
        return quotient ;
    }

    int multiply(int num1,int num2){
        int mul=0;
        for(int i=1;i<=num2;i++){
            mul=mul+num1;
        }
        return mul;
    }
    int add(int num1,int num2){
        return num1+num2;
    }
    int sub(int num1,int num2){
        return num1-num2;
    }
}
```

String

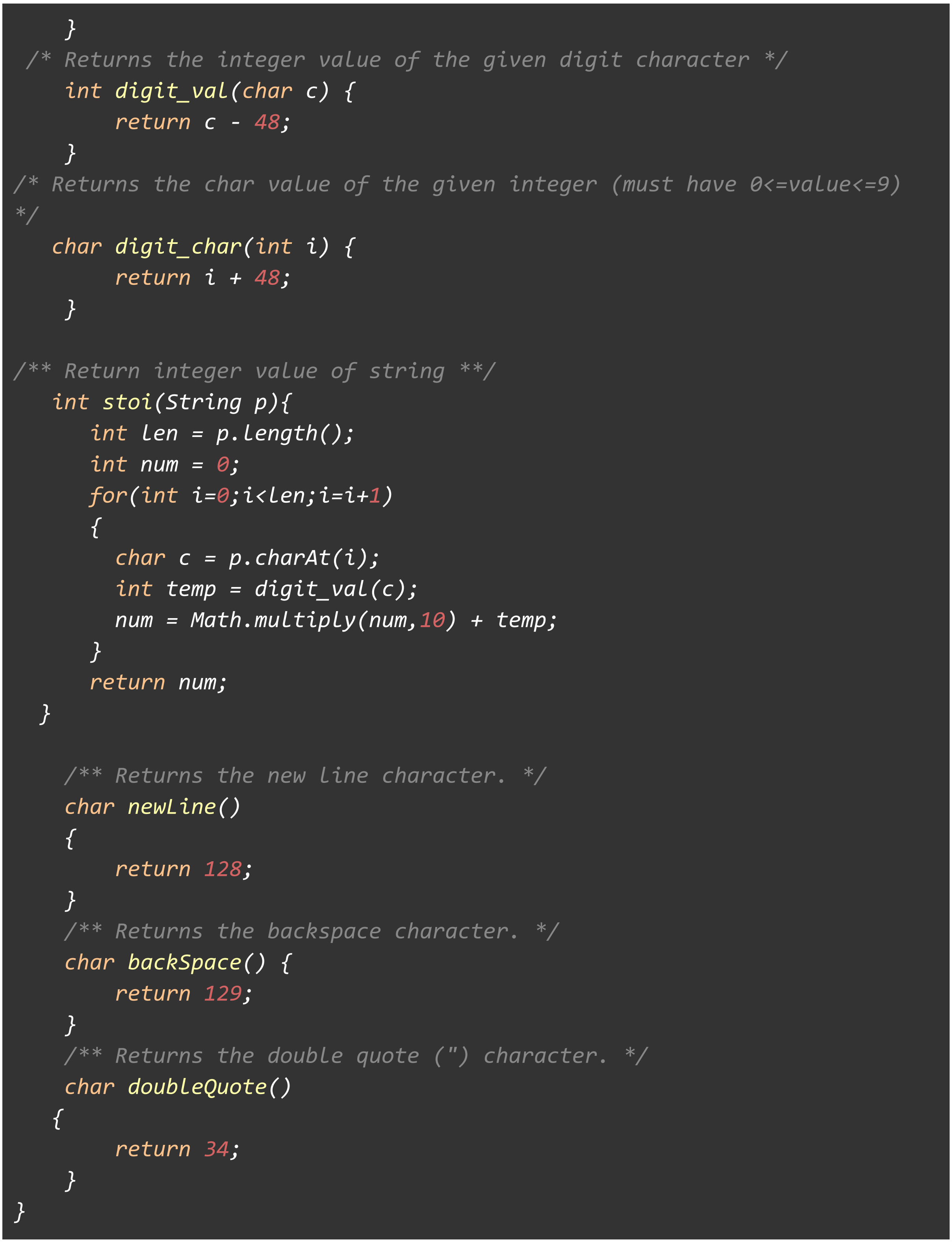
#include <String>

Note : While writing String Library we also need to include Math library for multiplication and division operations and Array.

```
#include <Math>
#include <Array>
class String{
    Array buffer;
    int buffer_len;
    int str_len;
    /* Constructs a new empty String with a maximum length of maxLength.
*/
    String (int maxLength) {
        if( maxLength = 0 ) {
            let maxLength = 1;
        }
        buffer = new Array(maxLength);
        buffer_len = maxLength;
        str_len = 0;
        return this;
    }
    /* De-allocates the string and frees its space. */
    void dispose()
   {
        Array.dispose(buffer);
        return;
    }
    /* Returns the current length of this String. */
    int length()
   {
        return str_len;
    }
    /* Returns the character at location j. */
    char charAt(int j)
    {
        return buffer[j];
    }
    /* Sets the jth character of this string to be c. */
   void setCharAt(int j, char c) {
        buffer[j] = c;
        return;
    }
    /* Appends the character c to the end of this String.
     *  Returns this string as the return value. */
   String appendChar(char c) {
        if( str_len < buffer_len ) {
            buffer[str_len] = c;
            str_len = str_len + 1;
        }
        return this;
    }
    /* Erases the last character from this String. */
   void eraseLastChar() {
        if( str_len > 0 ) {
            str_len = str_len - 1;
        }
        return;
    }
    /* Returns the integer value of this String until the first non
numeric character. */
     int intValue() {
        int int_val;
        int i;
        bool neg;

        int_val = 0;

        if( (str_len > 0) & (buffer[0] = 45) ) {
            // '-'
            neg = true;
            i = 1;
        }
        else {
            neg = false;
            i = 0;
        }
        while( (i < str_len) & String.is_digit(buffer[i]) ) {
            int_val = (int_val * 10) + String.digit_val(buffer[i]);
            i = i + 1;
        }
        if( neg ) {
            return -int_val;
        }
        else {
            return int_val;
        }
    }
  /* Returns whether the given char is a digit or not */
    bool is_digit(char c) {
        return ~(c < 48) & ~(c > 57);
```



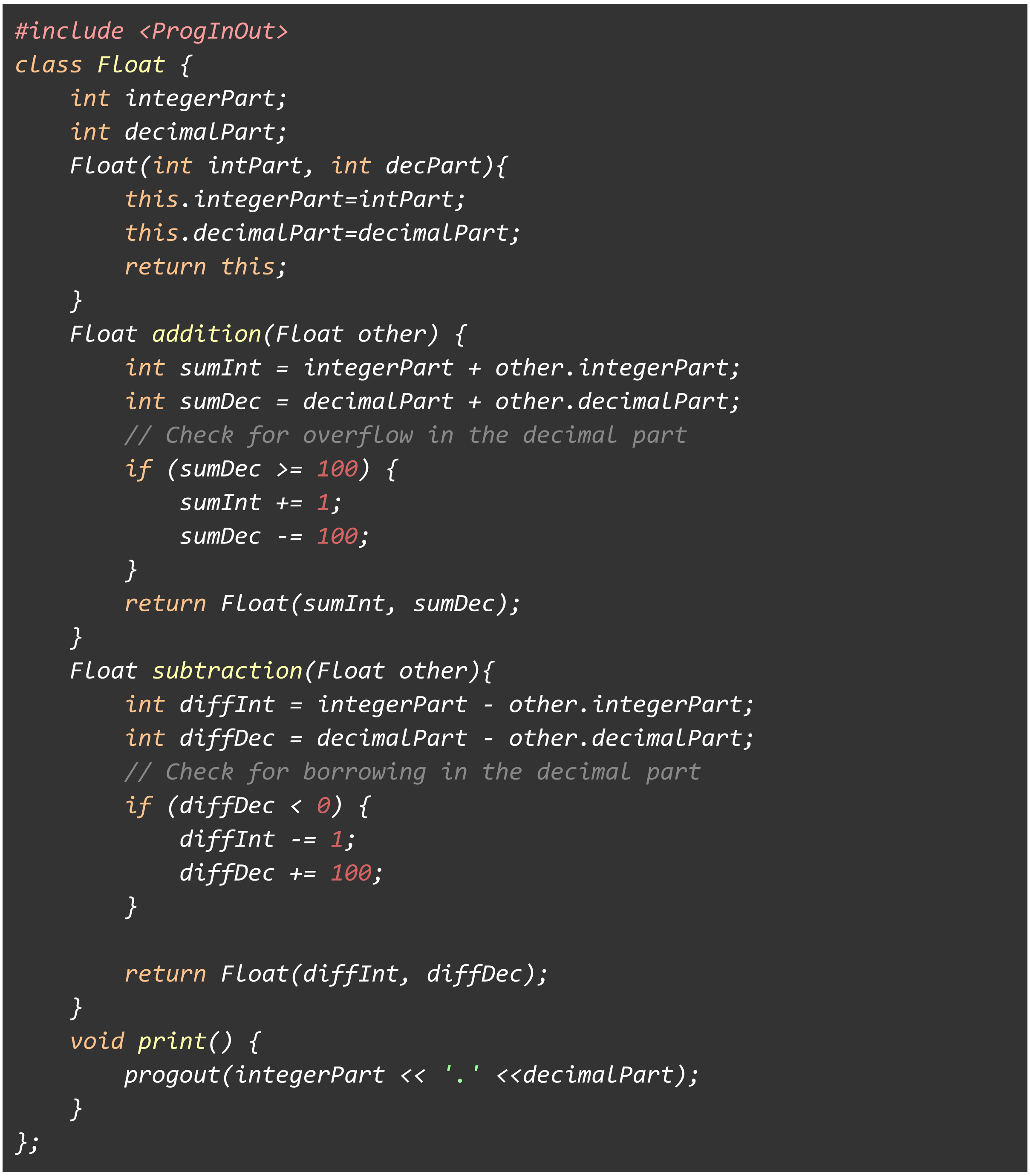
```
    }
 /* Returns the integer value of the given digit character */
    int digit_val(char c) {
        return c - 48;
    }
/* Returns the char value of the given integer (must have 0<=value<=9)
*/
   char digit_char(int i) {
        return i + 48;
    }

/** Return integer value of string **/
   int stoi(String p){
      int len = p.length();
      int num = 0;
      for(int i=0;i<len;i=i+1)
      {
        char c = p.charAt(i);
        int temp = digit_val(c);
        num = Math.multiply(num,10) + temp;
      }
      return num;
  }

    /** Returns the new line character. */
    char newLine()
    {
        return 128;
    }
    /** Returns the backspace character. */
    char backSpace() {
        return 129;
    }
    /** Returns the double quote (") character. */
    char doubleQuote()
   {
        return 34;
    }
}
```

Float

#include <Float>



```
#include <ProgInOut>
class Float {
    int integerPart;
    int decimalPart;
    Float(int intPart, int decPart){
        this.integerPart=intPart;
        this.decimalPart=decimalPart;
        return this;
    }
    Float addition(Float other) {
        int sumInt = integerPart + other.integerPart;
        int sumDec = decimalPart + other.decimalPart;
        // Check for overflow in the decimal part
        if (sumDec >= 100) {
            sumInt += 1;
            sumDec -= 100;
        }
        return Float(sumInt, sumDec);
    }
    Float subtraction(Float other){
        int diffInt = integerPart - other.integerPart;
        int diffDec = decimalPart - other.decimalPart;
        // Check for borrowing in the decimal part
        if (diffDec < 0) {
            diffInt -= 1;
            diffDec += 100;
        }

        return Float(diffInt, diffDec);
    }
    void print() {
        progout(integerPart << '.' <<decimalPart);
    }
};
```

MAC Instruction

multiply-accumulate (MAC) operation computes the product of two numbers and adds that product to an accumulator.

```
a ← a+(b*c)
```

How to define mac operation -

```
mac a; // for setting it to some value
mac reset; // resetting mr back to 0
int res=r1@r2 // for mac -> mr+(r1*r2)
```

it follows the convention that the first variable in a mac statement is treated as the accumulator followed by the other 2 operands.

The grammar rule for this is defined as like this

Simple Example of MAC operator

```
#include <ProgInOut>
int main(){
      // Declare variables
      int a = 5;
      int b = 2;
      int c = 3;

      // setting mac to value a mr=a
      mac a;
      // Use the '@' operator to compute mr=mr+(b*c)
      int result=b@c;

      // Display the result
      progout(result);  // Output: 11
      return 0;
}
```

Here the mac operator performed a multiply-accumulate operation on the variables a, b, and c.
The result is then printed, and the output is 11 because it calculates a += b * c.

Examples

Class Example

```
Class Student
{
      int id;
      int marks;
      Student(int id,int marks)
{
            this.id=id;
            this.marks=marks;
            return this;   // returns the RAM's base address of Student
class
}
      void displayID()
{
            progout(id);
      }
}
```

Example 1 - Easy 1

Array and For Loop

```
#include <ProgInOut>
int main(){
  int  arr[5] = {1,2,3,4,5};
  for(int i=0;i<5;i=i+1)
  {
      progout(arr[i]);
  }
}
```

Example 2 - Easy 2

Finding the inverse of the given number i.e inverse of 123 is 321

```
#include <Math>
#include <ProgInOut>
int inverse(int number)
{
      int temp = number;
      int r = 0;
      int res = 0;
      while( temp!=0)
      {
              r = temp%10;
              int temp1 = res*10;
              res = temp1 + r;
              temp = temp/10;
       }
       return res;
}
int main(){
     int number;
     progin(number);
     int inv = inverse(number);
     progout("The inverse is: ");
     progout(inv);
}
```

Example 3 - Easy 3

Finding greater number among two numbers

```
int sub(int a, int b)
{
    int  res = a-b;
    return res;
}

int main()
{
    int var1,var2;
    progout("Enter the variable1:  ");
    progin(var1);
    progout("Enter the variable2:  ");
    progin(var2);
   int res = sub(var1,var2);
 if ( res<0 ){
    progout("var2 is greater"); }
 else
 { progout("var1 is greater"); }
   return 0;
}
```

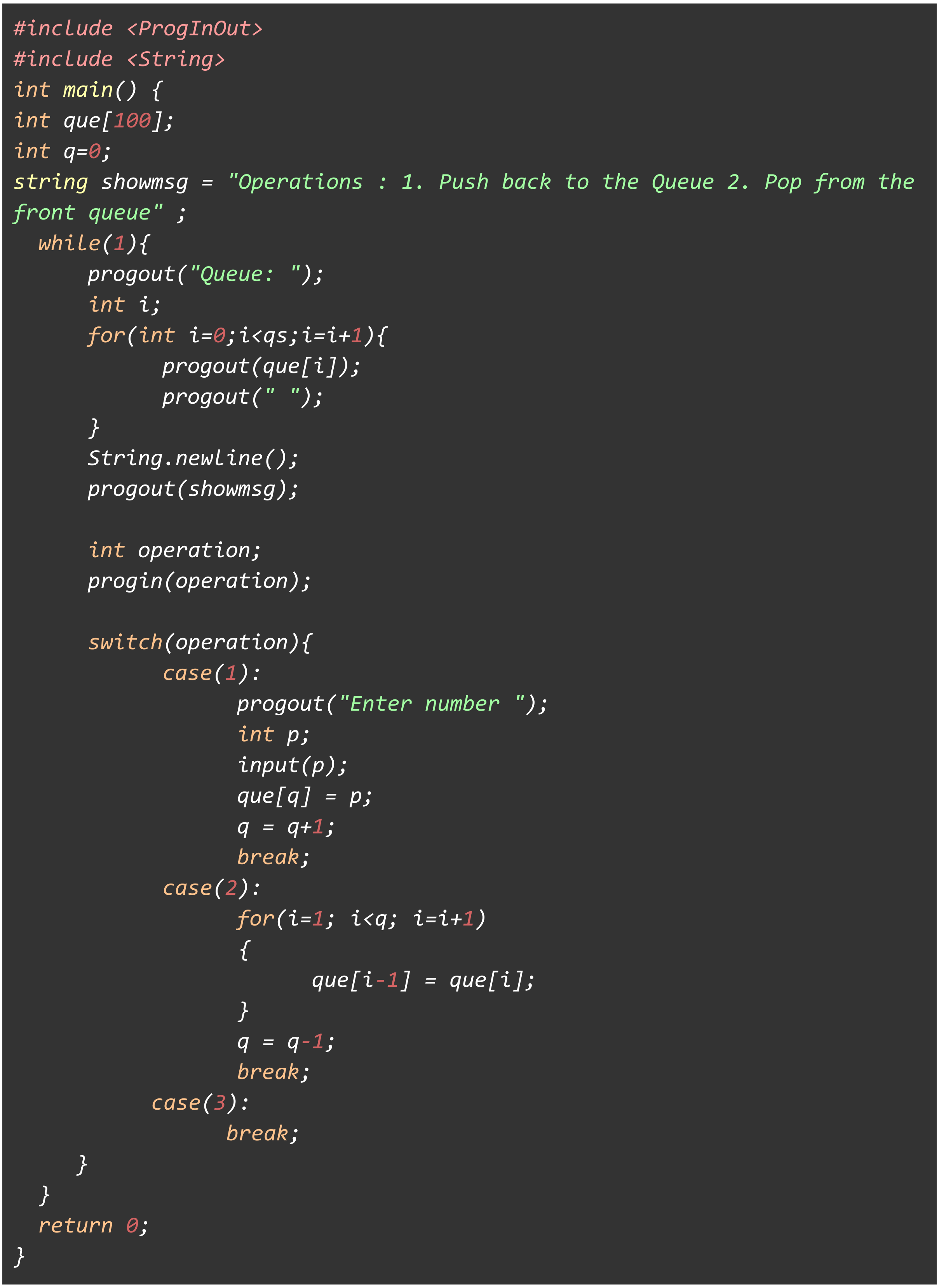
Example 4 - Easy 4

Various operations using switch-case along with three different libraries

```
#include <ProgInOut>
#include <Math>
#include <String>
int main()
{
    int a;
    int b;
    int c;
    progout("Enter a value = ");
  String.newline();
    progin(a);
    progout("Enter b value = ");
  String.newline();
    progin(b);
    progout("Select the operation: 1. Add(+) 2. Subtract (-) 3. Bitwise
And (&) 4. Multiply (*)");
  String.newline();
    int op;
    progin(op);
    progout("Result: ");
    switch(op){
        case(1): c=a+b;
                          break;
        case(2): c=a-b;
                          break;
        case(3): c=a&b;
                          break;
        case(4): c=a*b;
                          break;
        default: progout("Invalid");
            break;
    }
    progout(c);
    return 0;
}
```

Example 5 - Moderate 1

Queue - Push and Pop using array



```
#include <ProgInOut>
#include <String>
int main() {
int que[100];
int q=0;
string showmsg = "Operations : 1. Push back to the Queue 2. Pop from the
front queue" ;
  while(1){
      progout("Queue: ");
      int i;
      for(int i=0;i<qs;i=i+1){
            progout(que[i]);
            progout(" ");
      }
      String.newline();
      progout(showmsg);

      int operation;
      progin(operation);

      switch(operation){
            case(1):
                  progout("Enter number ");
                  int p;
                  input(p);
                  que[q] = p;
                  q = q+1;
                  break;
            case(2):
                  for(i=1; i<q; i=i+1)
                  {
                        que[i-1] = que[i];
                  }
                  q = q-1;
                  break;
           case(3):
                 break;
     }
  }
  return 0;
}
```

Example 6

Nth Fibonacci Number

```
#include <ProgInOut>
int fib(int n){
    if (n <= 1)
    { return n; }

    return fib(n - 1) + fib(n - 2);
}

int main()
{
    int n = 9;
    progout(n);
    progout("th Fibonacci Number: " );
    int res = fib(n);
    progout( res );
    return 0;
}
```

Example 7 - Matrix Multiplication - Hard 1

3x3 Matrix Multiplication

- Elements of a matrix are stored in 1D array as follows :
    - a[3][3] - [[a0, a1, a2], [a3, a4, a5], [a6, a7, a8]]
    - a[9]   - [a0, a1, a2, a3, a4, a5, a6, a7, a8, a9]

```
#include <ProgInOut>
#include <String>
#include <Math>
int main(){
      int a[9];
      int b[9];
      progout("Enter the contents of matrix 1: ");
      String.newLine();
      int i;
      for(i=0; i<9; i=i+1){
            int t;
            progin(t);
            a[i] = t;
      }
```

```
    progout("Enter the contents of matrix 2: ");
    String.newline();

    for(i=0; i<9; i=i+1){
        int t;
        progin(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;
            mac reset;
            for(k=0; k<3; k=k+1){
                int temp1 = a[row*3+k] ;
                int temp2 = b[k*3+col] ;
                // mr = mr+(temp1*temp2);
               sum =temp1@temp2;

            }
            int temp6 = Math.multiply(row,3)
            c[temp6+col] = sum;
        }
    }
    for(i=0; i<9; i=i+1){
        progout(c[i]);
        if((i+1)%3 == 0){
            String.newline();
        }
        else{
            progout(" ");
        }
    }
    return 0;
}
```

Example 8 - Moderate 2

Selection Sort

```
#include <ProgInOut>
#include <String>
int  main(){
    int arr[10];
    progout("Enter array elements: ");
    int i;
    for(i=0; i<10; i=i+1){
        progin(arr[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;
    }
    progout("Sorted array: ");
    for(i=0; i<10; i=i+1)
    {
        progout(arr[i]);
        progout(" ");
    }
    return 0;
 }
```

Example 9

To check if a number is prime or not

```
#include <ProgInOut>
#include <Math>

bool isPrime(int n)
{
    // Corner case
    if (n <= 1)
    { return false; }

    // Check from 2 to n-1
  int l = Math.division(n,2);
    for (int i = 2; i <= l; i++)
      {   if (n % i == 0)
            { return false; }
            return true;
   }
}
// Driver code
int main()
{
  int num;
  progout("Enter the number : ");
  progin(num);
  bool stat = isPrime(num);
  if( stat )
  {
    progout("Number is prime");
  }
  else
  {
    progout("Number is composite");
  }
    return 0;
}
```

Example 10

Concatenate two strings

```
#include <ProgInOut>
#include <String>

int main()
{
      String s1("Hello");
      String s2("World");

      // s1 + s2
      int l2 = s2.length();
      for(int i=0; i<l2;i=i+1)
      {
            char temp = s2.charAt(i);
            s1.appendChar(temp);
      }
      progout("The Concatenated String is:  ");
      progout(s1);
return 0;
}
```

Example 11 - Float

```
#include <Float>
int main() {
    // Example usage
    Float float1(3, 50); // 3.50
    Float float2(2, 60); //2.60

    Float result_add = float1.addition(float2);
    Float result_sub = float1.subtraction(float2);

    progout( "Result of Addition: ");
    result.print(result_add); // 6.10
    progout( "Result of Subtraction: ");
    result.print(result_sub); // 0.90
    return 0;
}
```

Example 12 - BST Insert - Hard 2

```
#include <String>
#include <ProgInOut>
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
    progout("Tree: ");
    String.newLine();
    String.newLine();
    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){
            progout(tree[m+i]);
            progout(" ");
        }
        String.newLine();
        m = m + n;
        n = n * 2;
    }
    String.newLine();
    String.newLine();
    // ------------------
    progout("Insert: ");
    int in;
    progin(in);
    int cur = 0;
    while(tree[cur] != 0){
    if(in < tree[cur]){
        cur = 2*cur + 1;
    }
    else
    {
        cur = 2*cur + 2;
    }
    }
    tree[cur] = in;
    }
}
```

MAC

```
#include <ProgInOut>
int main(){
    int a[3]={1,2,3};
    int b[3]={1,2,3};
    mac reset;
    int product=0;
    for(int i=0;i<3;i++){
        product=a[i]@b[i];
    }
    progout("Dot Product: ");
    progout(product);
    return 0;
}
```