C++ Part 2: Control Statements, Arrays and Strings, Dynamic Memory

Control Statements

- Conditional Control Statements
  - ➢ Branching Statements
  - ➢ Looping Statements
- Unconditional Control Statements

Branching Statements

- The IF selection control statement
- The IF..ELSE selection control statement
- ELSE..IF multiple selection statement
- Nested if..else
- SWITCH selection control statement
- Conditional Operator (?:)

The IF Selection Statement

SYNTAX:

```
if(condition)
 {
statement1;
statement2;
 }
```

Example:

```
 int grade;
cout <<”Enter your marks: “;
cin >> grade;
if (grade >= 60)
 cout <<”Passed”<< endl;
```

The IF..ELSE Selection Statement

SYNTAX

```
if(condition)
{
statement1;
statement2;
}
Else
{
statement3;
statement4;
}
```

Example:

```
if (grade >= 60)
cout <<"Passed"<< endl;
else
cout <<"Failed"<< endl;
```

ELSE..IF Multiple Selection Statement

SYNTAX:

```
if(condition1)
{
 statement1;
}
else if(condition2)
{
 statement2;
}
else
{
 statement3;
}
```

Example:

```
if (grade >= 90) // 90 and above gets "A"
cout <<"A"<< endl;
else if (grade >= 80) // 80-89 gets "B"
cout <<"B"<< endl;
else if (grade >= 70) // 70-79 gets "C"
 cout <<"C"<< endl;
else if (grade >= 60) // 60-69 gets "D"
 cout <<"D"<< endl;
else   // less than 60 gets "F"
 cout <<"F"<< endl;
```

Nested If..Else

Syntax:

```
if ( condition 1 )
{
  if ( condition 2 )
            statement1;
  else
            statement2;
}
else
{
  if ( condition 3 )
            statement3;
  else
            statement4;
}
```

Example:

```
if (x > 5)
{
    if (y > 5)
            cout <<"x and y are >
5"<< endl;
    else
            cout <<"y is <= 5"<<
endl;
}
else
{
    cout <<"x is <=5"<< endl;
}
```

SWITCH Selection Statement

SYNTAX

```
switch( x )
{
  case x1:
            statements1;
            break;
  case x2:
            statements2;
            break;
  case x3:
            statements3;
            break;
  default:
            statements4;
            break;
}
```

SWITCH Selection Statement

Example:

```
switch (operation)
{
  case'+':
          result = a + b;
          break;
  case'-':
          result = a - b;
          break;
  case'*':
          result = a * b;
          break;
  case'/':
          result = a / b;
          break;
  default:
          cout <<"Invalid operation.
Program terminated."<< endl;
          return -1;
}
```

Conditional Operator (?:)

Syntax:

condition ? Action when true : action when false;

Example:

```
cout << (grade >= 60 ? “Passed” : “Failed”) << endl;
```

Looping Statements

- While loop
- Do..While Loop
- For loop

While loop

SYNTAX:

```
while( condition )
{
  statements;
}
```

Example:

```
int n = 10;
while (n>0)
{
  cout << n <<", ";
  --n;
}
```

Do..While Loop

SYNTAX:

```
do
{
    statements;
} while(condition);
```

Example:

```
do
{
  cout << count;
  count--;
} while (count <= 0);
```

For Loop

SYNTAX:

```
for (initialization; condition;
iteration)
{
 statements;
}
```

Example:

```
for (int n = 10; n>0; n--)
{
  cout << n <<", ";
}
```

UNCONDITIONAL CONTROL STATEMENTS

- The BREAK statement
- The CONTINUE statement
- The GOTO statement

The Break Statement

SYNTAX:

```
While  (condition)
{
  -----
  break;
  -----
}
```

Example:

```
for (int n = 10; n>0; n--)
{
   cout << n <<”, “;
   if (n == 3)
    {
       cout <<”countdown aborted!”<< endl;
       break;
    }
}
```

The Continue Statement

SYNTAX:

```
while(condition)
{
 ------
 continue;
 -----
}
```

Example:

```
for (int n = 10; n>0; n--)
{
 if (n == 5)
      continue;
 cout << n <<", ";
 }
```

The Goto Statement

SYNTAX:

```
-----
-----
goto abc;
-----
-----
abc: -----
-----
```

Example:

```
int n = 10;
mylabel:
cout << n <<", ";
 n--;
 if (n>0)
 goto mylabel;
```

Arrays

- A sequence of variables that can store value of one particular data type.
- SYNTAX:

type array_name[size];

- Examples:
  - Array initialization during declaration

```
int test[5] = {12, 3, 4, -3, 9};
int test[] = {12, 3, 4, -3, 9};
```

Example Program- Array

```
/*Program to store 5 numbers entered by user in an array and display first and last number only.*/
#include <iostream>
using namespace std;
int main()
{
  int n[5];
  cout <<"Enter 5 numbers: ";
/*  Storing 5 number entered by user in an array using for loop.
*/
 for (int i = 0; i < 5; ++i)
{
 cin >> n[i];
}
cout <<"First number: "<< n[0]
<< endl; // first element of an
array is n[0]
cout <<"Last number: "<< n[4]
<< endl; // last element of an
array is n[SIZE_OF_ARRAY - 1]
 return 0;
 }
```

Multi-Dimensional Array

Two-dimensional arrays have rows and columns.

Example:

```
int a[2][2];
```

evaluates to,

```
 a[0][0] =2;  a[0][1] = 3;
a[1][0] =4;  a[1][1] =5;
```

Example:

```
char ticTacToeBoard[3][3] =
{
{'x', 'x', 'o'},
{'o', 'o', 'x'},
{'x', 'o', ' '}
 };
```

```
/* for loop to print contents
of a 2 dimensional array */

for (i = 0; i < 4; i++)
{
  for (j = 0; j < 4; j++)
  {
      a[i][j] = k;
      cout << a[i][j] <<'\n';
      k++;
   }
 }
```

Passing Array Argument To Function

- Function Declaration:

Type function_name(type array[])

- Function Call
  - Declare the array

  Type array_name[size];

  - Pass array as parameter

  Function_name(array_name)

```
#include <iostream>
using namespace std;
void printfunc(int my_arg[], int i)
{
 for (int n = 0; n < i; n++)
  cout <<my_arg[n] <<'\n';
}
int main()
{
 int my_array[] = { 1, 2, 3, 4, 5 };
 printfunc(my_array, 5);
 return 0;
}
```

Strings

- objects that represent sequences of characters
- two types of strings
    - C-strings

```
char str[] = “C++”;
char str[4] = “C++”;
char str[] = {‘C’,’+’,’+’,’\0’};
char str[4] = {‘C’,’+’,’+’,’\0’};
```

    - The Standard C++ Library String Class

```
#include <iostream>
using namespace std;
int main()
{
 char str[100];
 cout <<”Enter a string: “;
 cin.get(str, 100);
 cout <<”You entered: “<< str << endl;
 return 0;
 }
```

S.N.	Function & Purpose
1	strcpy(s1, s2); Copies string s2 into string s1.
2	strcat(s1, s2); Concatenates string s2 onto the end of string s1.
3	strlen(s1); Returns the length of string s1.
4	strcmp(s1, s2); Returns 0 if s1 and s2 are the same; less than 0 if s1<s2; greater than 0 if s1>s2.
5	strchr(s1, ch); Returns a pointer to the first occurrence of character ch in string s1.
6	strstr(s1, s2); Returns a pointer to the first occurrence of string s2 in string s1.

The Standard C++ Library String Class

- Header

```
#include <string>
using namespace std;    // Or
using std::string;
```

- Declaration:

```
string name;
```

- Example:

```
string hello = "Hello, World";
```

```
#include <iostream>
 #include <string>
 using namespace std;
 int main()
{
 string full_name;
 cout <<"Hello, What is your
name? ";
 getline(cin, full_name);
 cout <<"Pleased to meet you,
"<< full_name << endl;
 }
```

String Manipulation Operations

- Concatenation: ‘+’ operator

string full_name = forename +” “+ surname;

- Concatenate strings one-at-a-time: ‘+=’

full_name +=”Joe “;

full_name +=”Bloggs”;

- Comparisons: Relational Operators

if (passwd ==”xyzzy”)

- Character sequences: subscript operator []

string hello = “Hello, World!”;

cout <<hello[0]<< endl;

String Manipulation Operations

- Search: find(), rfind()

```
 string.find(string pattern, int position);
```

Example:

```
getline(cin, input, '\n');
for (i = input.find("cat", 0); i != string::npos; i = input.find("cat", i))
 {
  cat_appearances++;
  i++;  /*Move past the last discovered instance to avoid finding same
string*/
}
```

String Manipulation Operations

- Substrings:

String. substr(int position, int length);

- String.erase(position, no_of_chars)

string text = “This is a test”;

text.erase(5, 5);

String Manipulation Operations

- String.replace(position, no_of_chars, text)

```
string text = “This is a test”;
text.replace(5, 2, “was”);
```

- String.insert(position, text)

```
string my_string = “ade”;
my_string.insert(1, “bc”);
```

String Manipulation Operations

- String.length()

```
string my_string;
len = my_string.length(); // or .size()
```

- String[]

```
string hello = “Hello, World!”;
 hello[0] = ‘*’;
```

Reference and Dereference operators

- & - address-of operator (reference operator)

  read as “address of”
- * - dereference operator,

  read as “value pointed to by”

Example:

foo = &myvar; //assigns the address of variable myvar to foo

bar = *foo; //assigns the value in the address space pointed to by foo, to bar.

DYNAMIC MEMORY

- Dynamic memory is allocated using operator *new*.
- It returns a pointer to the beginning of the new block of memory allocated.
- Synatx:

pointer = new type // allocate memory to contain one single element

pointer = new type [number_of_elements] // used to allocate a block (an array) of elements

- Example:

```
int * foo;
foo = new int[5];
```

Delete Operator

- Syntax:

delete pointer;

delete[] pointer;

- Used to free dynamically allocated memory.

Exercises

- Write a program to print out the multiplication table.
- Write a program that reads a character and prints out whether or not it is a vowel or a consonant.
- Write a function that returns the maximum value of an array of numbers.
- Write a program that takes a series of numbers and counts the number of positive and negative values.
- Write a program to compute the total resistance for any number of parallel resistors.
- Write a function “begins (string1, string2)” that returns true if string1 begins string2.