Chapter 2 The Basics

CS185

C++ program overview

Example 1: The smallest C++ program

```
int main(void)
{
    return 0;
}
```

The main function is the entry point of a C++ program. All standard C++ programs start by executing the content of the main function.

However, this program only returns the value 0 to the OS.
Facts:

- An open and closed parentheses are placed after the function name.
- Parentheses are used to hold the function arguments.
- The type of the value returned by the function is specified before the function name.
- void is a C++ type specifying a typeless type.
- The body of the function is written within the function block specified by the open and closed curly braces.
- The C++ programming language is a free format language. That doesn't mean though that you shouldn't write clean and properly organized code.

Example 2: Displaying a message

```
#include <iostream>

int main(void)
{
	std::cout << "Hello world!" << std::endl;

	system("pause");
	return 0;
}
```

Facts:

- ***cout*** is an output stream object provided by the Input Output Stream Library.
- The Input Output Stream Library file is ***iostream***.
- The ***#include*** directive instructs the C++ compiler to add the contents of an include file into the program during compilation.
- Library files are included within **<** and **>**.
- In our example, we are using the ***std::cout*** object in order to output ***Hello World!*** in the console window. The ***cout*** object uses the output operator **<<** in order to print things in the consle window.
- In C++, a string is enclosed between double quotation marks "".
- ***std::endl*** represents a new line.
- The semicolon at the end of the statement specifies the end of the statement.
- The ***system("pause");*** statement will make the console wait until we press any key in order to continue executing the remaining commands in the program. Without it the program will execute so fast that you will only see a console open and close.
- The program starts by executing the first statement and ends after executing the last statement.

Example 3: Comments

```
/* including the input/output stream library file,
   in order to be able to use the cout object and
   the system function*/
#include <iostream>


int main(void)
{
     //Printing to the console: Hello world!
     std::cout << "Hello world!" << std::endl;

     //make the console wait for a key press before resuming
     system("pause");
     return 0;
}
```

Facts:

- Comments are used to clarify and explain the code.
- Multiline comments start with a **/*** and end with a ***/.** The comments could be placed partially on a line, on an entire line, or on more than one line.
- Another way to have a single line comment is to start the line with a **//**
- The compiler ignores all the comments. In other words, the comments will not be present in the executable code.
- Therefore, comments have no effect on the program.

Example 4: Multiline instruction program

Code

```cpp
#include <iostream>

/* function prototype or declaration */
void DisplayHello(void);
void PlaceNewLine(void);
void PlaceSpace(void);
void DisplayWorld(void);

int main(void)
{
      DisplayHello(); PlaceSpace();
      DisplayWorld(); PlaceNewLine();
      DisplayHello(); PlaceNewLine();
      DisplayWorld(); PlaceNewLine();

      system("pause");
      return 0;
}

/* Function definition */
void DisplayHello(void)
{
      std::cout << "Hello";
}

void PlaceSpace(void)
{
      std::cout << " ";
}

void PlaceNewLine(void)
{
      std::cout << "\n";
}

void DisplayWorld(void)
{
      std::cout << "World";
}
```

Output

```
Hello World
Hello
World
```

Facts:

- The program contains four user functions.
- Functions need to be declared before being called.
- Functions need to be defined before being used or executed.
- The main function contains six function calls or six statements.
- The **"\n"** specifies the new line.
- User functions are declared outside the main function.
- Functions can be called many times.
- The program starts by executing the first instruction in main which is the DisplayHello function
- When the DisplayHello function is called, the execution flow changes to the first instruction within the function definition DisplayHello.
- When the last instruction (which is also the first instruction) of the function DisplayHello is executed, the execution returns to the instruction right after the function call, which is PlaceSpace.
- In reality, this is the flow of the executed instructions:
    - main
    - DisplayHello( );
    - std::cout << "Hello"
    - PlaceSpace( );
    - std::cout << " "
    - DisplayWorld( );
    - std::cout << "World"
    - PlaceNewLine( );
    - std::cout << "\n"
    - DisplayHello( );
    - std::cout << "Hello"
    - PlaceNewLine( );
    - std::cout << "\n"
    - DisplayWorld( );
    - std::cout << "World"

Example 5: Functions with arguments returning a value

Code

```
#include <iostream>

/* function prototype or declaration */
int Add(int, int);

int main(void)
{
        std::cout << Add(3, 5) << std::endl;

        system("pause");
        return 0;
}

/* Function definition */
int Add(int x, int y)
{
        return x + y;
}
```

Output

```
8
```

Facts:

- ***int*** is a C++ type (integer) specifying whole or integral numbers.
- ***int x*** and ***int y*** are called parameters.
- The function prototype specifies that the function takes two integer arguments and returns an integer.
- When a function has more than one parameter, a **comma** (,) is used to separate the parameters.
- The function definition should have the same number and parameter types.
- When a function with parameters is called, the arguments are received as parameters by the function where the function's instructions are specified.
- When arguments are passed to the parameters, the order of the parameters is respected. In our case, x would be equal to 3 and y would be equal to 5.
- The last statement of the function definition returns the result of the arithmetic expression: ***x+y***.
- **x** and **y** are called variables. A variable is a name assigned to a data storage location.
- In C++, a variable must be defined before it can be used. A variable definition specifies its name and type.
- The compiler uses the type in order to know how much memory to allocate.

Example 6: Variables as arguments

Code

```
#include <iostream>

/* function prototype or declaration */
int Add(int, int);

int main(void)
{
       int i = 3, j = 5;
       std::cout << Add(3, 5) << std::endl;

       system("pause");
       return 0;
}

/* Function definition */
int Add(int x, int y)
{
       return x + y;
}
```

Output

```
8
```

Facts:

- Two integer variables ***i*** and ***j*** are declared and defined.
- By declaration, we mean that the rest of the function ***main*** knows about the presence and type of ***i*** and ***j***.
- In other words, the scope and type of ***i*** and ***j*** is within the body of function ***main***.
- Then, an assignment operator is used in order to assign the value 3 and 5 to ***i*** and ***j*** respectively.
- The function ***Add*** is used by having two variables as arguments while in the previous example the arguments were constants.

Example 7: User input

Code

```
#include <iostream>

/* function prototype or declaration */
int Add(int, int);

int main(void)
{
       int i, j;
       std::cout << "Enter two integer values: ";
       std::cin >> i;
       std::cin >> j;
       std::cout << Add(i, j) << std::endl;

       system("pause");
       return 0;
}

/* Function definition */
int Add(int x, int y)
{
       return x + y;
}
```

Output

```
3
5
8
```

Facts:

- Instead of using the assignment operator to assign value to the variables, the input object ***std::cin*** is used.
- ***std::cin*** is defined in the input output stream library ***<iostream>***.
- ***std::cin*** will read the value from the console (user input) and assign it to the specified variable.
- **std::cin >> i;** get the value from the console and store it in ***i***.

Example 8: Conditional Expression

Code

```
#include <iostream>

int main(void)
{
    int i;

    std::cout << "Enter a number then press return: ";
    std::cin >> i;

    if (i > 100)
    {
       std::cout << i << " is larger then 100" << std::endl;
    }
    else
    {
       std::cout << i << " is less or equal then 100" << std::endl;
    }

    system("pause");
    return 0;
}
```

Facts:

- ***i > 100*** is the Boolean expression that evaluates to ***true*** or ***false***.
- If the expression is ***true***, then the statement (or the block of statements enclosed between an opening and a closing curly brace) following the condition, will be executed.
- If the expression is ***false***, then the statement following the condition will be skipped.
- The statement following the else is executed only when the conditional expression is false.
- In other words, only one of the ***std::cout*** statements will be executed.
- The conditional statement starts with the keyword ***if (*** , followed by an opening parenthesis, followed by a Boolean expression, followed by a closing parenthesis ***)***, followed by an instruction.

```
if (Boolean expression)
{
   instruction;
}
```

- Notice that there is no semicolon after the closing parenthesis of the Boolean expression because the conditional statement has not ended yet.

Example 9: Loops

Code

```
#include <iostream>

int main(void)
{
        int i;

        for (i = 0; i < 10; i = i + 1)
        {
                std::cout << i << " ";
        }

        std::cout << std::endl;

        i = 0;
        while (i < 10)
        {
                std::cout << i << " ";
                i = i + 1;
        }

        system("pause");
        return 0;
}
```

Output

```
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
```

Facts:

- The for loop form is:
  for (expression1; expression2; expression3)
  {
  statement;
  }
- The loop is initialized through expression1: ***i=0;***
- Expression2 specifies the test made before each iteration: ***i<10;***
- If expression2 is true then
  - The statement ***std::cout << i << " ";*** is executed
  - Then expression3: ***i=i+1*** is executed
  - The loop iterates until expression2 is false.

- If expression2 is false,
    - The for loop will exit
    - And the control is transferred to the statement following the statement.
      std::cout << std::endl;
    - Expression3 is evaluated after each iteration.
- Any or all of the three for expressions may be omitted, but the semicolon must remain.

```
for (   ;   ;   )
{
   statement;
}
```

- The while loop form is:

```
while(expression)
{
    statement
}
```

- If the expression ***i < 10*** is ***true***, the statement is executed until the expression becomes ***false***.
- In our case the statement is made from a block enclosed between curly braces:

```
{
   std::cout << i << " ";
   i = i+1;
}
```

- If expression is ***false***, the execution resumes at the following statement. In our case, the following statement is the end of the program.
- The expression is evaluated before the statement is executed.
- When the expression is ***false*** from the first time, the statement will never be executed.

Example 10: One Dimensional Array

Code

```
#include <iostream>

int main(void)
{
	int i, A[10];

	for (i = 0; i < 10; i = i + 1)
	{
		A[i] = i;
	}

	for (i = 0; i < 10; i = i + 1)
	{
		std::cout << A[i] << " ";
	}

	system("pause");
	return 0;
}
```

Output

```
0 1 2 3 4 5 6 7 8 9
```

Facts:

- A one-dimensional array is a collection of variables of the same type that are referred to by the same name.
- In our case int ***A[10];*** reserved 10 integers.
- Array elements are accessed through an index, in our case the index is ***i***.
- The first element is accessed by index 0, ***A[0]***.
- The highest address corresponds to the last element and it is accessed by index **(total number of elements – 1)**, in our case it is ***A[9]***.
- The amount of storage required to hold an array depends on the type and the total number of elements, in our case it is **10 * 4 = 40**, since each integer is 4 bytes.
- The C++ compiler does not perform index range checking. In other words, if you try to index things outside of the array the compiler will not stop you.
- The array element is accessed by indexing the array name.
- It is done by writing the index enclosed between brackets placed after the array name (arrayName[index]): ***A[i] = i;***

Example 11: Pointers

Code

```
#include <iostream>

int main(void)
{
        int i = 5;
        int *pi;

        pi = &i;
        std::cout <<  pi << std::endl;
        std::cout << *pi << std::endl;

        system("pause");
        return 0;
}
```

Output

```
001EF824
5
```

Facts:

- The program declared two variables, one of type integer: ***int i=5;*** and one of type pointer to integer: ***int *pi;***
- A pointer variable holds the ***address*** of an integer.
- Since ***&i*** is the address of ***i***, then the assignment expression ***pi=&i;*** will assign the address of ***i*** to the variable ***pi***.
- If you display the content of variable ***pi***, you will get the address of ***i*** (*in our case 001EF824).*

  PS: It might be different every run. It depnds where the variable ***i*** got created.
- If you want to display the content of the address stored in ***pi***, then you need to access the content of the address using the dereference operator ****pi***.
- Recap:
  pi = address of i.
  *pi = content of the address stored in pi, in our case it is 5.

Example 12: Structure

Code

```
#include <iostream>
#include<math.h>

struct point
{
        int x;
        int y;
};

/* function prototype or declaration */
double Distance(point, point);


int main(void)
{
        point p1, p2;
        p1.x = 2; p1.y = 1;
        p2.x = 7; p2.y = 3;

        std::cout << Distance(p1, p2) << std::endl;


        system("pause");
        return 0;
}

/* Function definition */
double Distance(point p1, point p2)
{
        int deltaX, deltaY;
        deltaX = p2.x - p1.x;
        deltaY = p2.y - p1.y;

        return sqrt((deltaX * deltaX) + (deltaY * deltaY));
}
```

Output

```
5.385165
```

Facts:

- A structure is an object consisting of a sequence of named members of various types.
- It is a collection of variables referenced under one name.
- The collection of variables is logically related.
- It provides a convenient way to keep related information together.

- A structure is declared by typing the keyword struct, followed by the structure name, followed by the structure members, enclosed between curly braces, for example:

```
struct point
{
   int x, y;
};
```

- Once a structure is declared, variables having the structure type could be declared, by typing the structure name, followed by the variable name, for example:

```
point p2;
```

- The dot "**.**" operator is used to access the structure member.
- First write the structure variable name, followed by a dot, followed by a member, for example:

```
deltaX = p2.x - p1.x;
```