

Modern C++

An effective short way

By
Mustapha Ossama Abdelhalim
2024

Contents

1	Starter and Installation	3
1.1	For windows	3
2	Basics	5
2.1	Introduction	5
2.2	Hello World	7
2.3	Types, Variables, and Arithmetic.....	Error! Bookmark not defined.
2.3.1	Types	Error! Bookmark not defined.

1 Starter and Installation

Modern C++ starts with C11, this book introduce C11 and later on, the moving to C17 section

1.1 For windows

- 1- Go to winlibs.com
- 2- Determine which list you will choose from UCRT runtime if you are using windows 10 or 11, or choose MSVCRT runtime if you are using older versions of windows.
- 3- If you will use the gcc for application that runs only on windows choose MCF threads, if you are using application that runs on windows and later maybe used on Linux distribution; choose POSIX threads

I will choose Win64 in UCRT runtime in POSIX thread section as I have windows 10 x64 and have 7zip installed see Figure 1 gcc releases

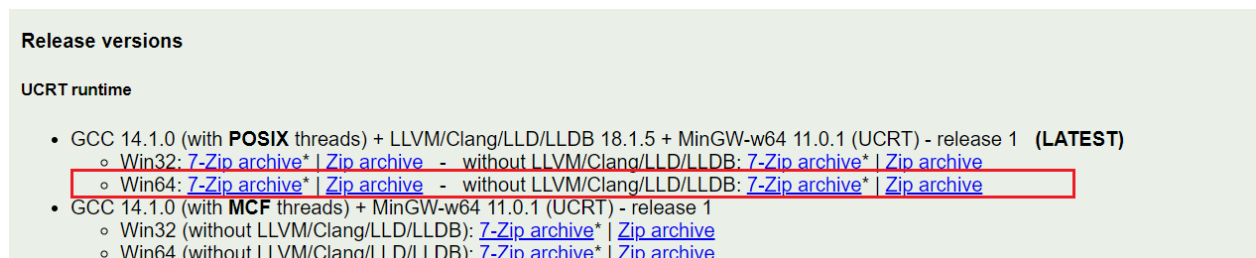


Figure 1 gcc releases

See this video for more details [LINK](#)

After downloading and extracting, move the mingw to c directory and get the bin path in environment variable and make sure to delete the old gcc from environment variables if exists. See Figure 2 adding bin folder path to environment variables

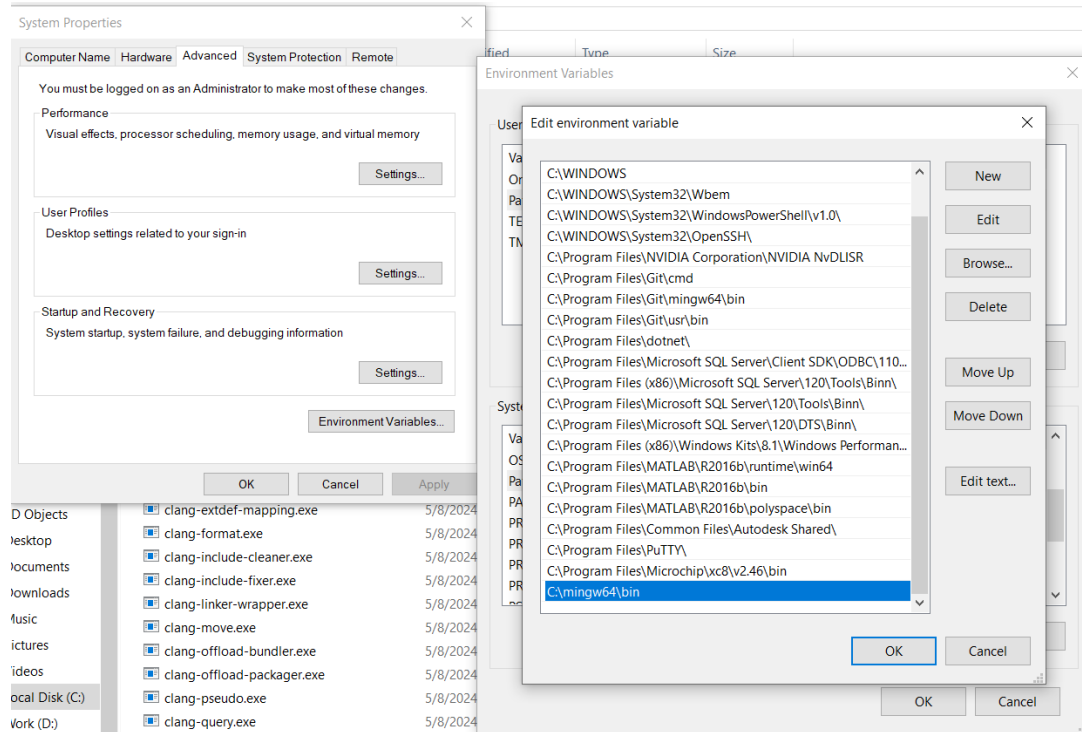


Figure 2 adding bin folder path to environment variables

Type in cmd gcc --version and you should see that gcc installed see Figure 3 verifying gcc installation

```
C:\Users\Mustapha>gcc --version
Microsoft Windows [Version 10.0.19045.4529]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Mustapha>gcc --version
gcc (MinGW-W64 x86_64-ucrt-posix-seh, built by Brecht Sanders, r1) 14.1.0
Copyright (C) 2024 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

Figure 3 verifying gcc installation

2 Basics

In this chapter, the Basics of C++ will be introduced as a refresher, the following topics will be introduce:

- **First program**
 - Compilation Hello World
- **Variables and Data Types**
 - Primitive types: int, char, float, double, bool
 - Derived types: arrays, pointers, references
 - User-defined types: structs, enums, classes
- **Operators and Expressions**
 - Arithmetic operators: +, -, *, /, %
 - Relational operators: ==, !=, >, <, >=, <=
 - Logical operators: &&, ||, !
 - Bitwise operators: &, |, ^, ~, <<, >>
 - Assignment operators: =, +=, -=, *=, /=, %=, &=, |=, ^=, <<=, >>=
 - Increment and decrement operators: ++, --
 - Conditional operator: ?:
- **Control Structures**
 - Conditional statements: if, if-else, nested if, switch-case
 - Looping statements: for, while, do-while
 - Jump statements: break, continue, goto, return

2.1 Introduction

A **programming language** is set of instruction to perform a task, that's it

In this book we will use notepad++ (even the simple preinstalled notepad will work fine) and compile our program in command prompt CMD, also its completely fine to use any integrated development environment (IDE), but make sure that you are using C11 gcc version.

C++ language has two types of files headers files(.h files) and source files (.cpp files), to compile the program and make it executable for windows (aka converted to .exe files to run on windows). you will use the following command in cmd

```
g++ -std=c++11 name.cpp -o name.exe
```

let's break it down

- **g++** is the gcc command to perform compilation
- **-std-c++11** is flag to specify the version of c11
- **name.cpp** is our source file
- **-o** is the flag for output the .exe file
- **name.exe** is the name of output

2.2 Hello World

Lets compile our first program !

```
#include<iostream>
int main() {
    std::cout<<"Hello World";
    return 0;
}
```

- `#include<iostream>`

is library that permit us to output data and take input from user

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

Is the entry point for our program, all programs and applications should have that function (later functions will be expressed)

- `std::cout<<"hello world";`

is the command to output hello world on the screen

- 1- make a file named Hello.cpp for example
- 2- type the code above
- 3- open cmd in the same directory as the file Hello.cpp
- 4- type: `g++ -std=c++11 Hello.cpp -o Hello.exe`
- 5- to run the program type: `Hello.exe`

the output should be as follows in Figure 4 first program

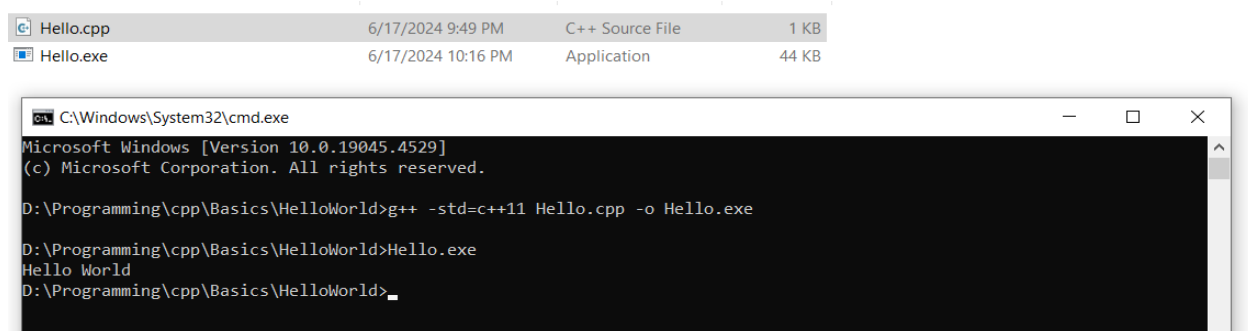


Figure 4 first program

2.3 Variables and data types

C++ has types to declare each variable, each variable should have a keyword to define if it integer (like 10, 99, and 120) or decimal aka float like (10.2, 0.2, and 22.8) or character (like 'a', 'b' and 'c'), this declaration specifies:

- How the variable is stored in memory and takes how much of program memory
- How operations change that variable

The types in C++ are as follows in Figure 5 Types in C++ :

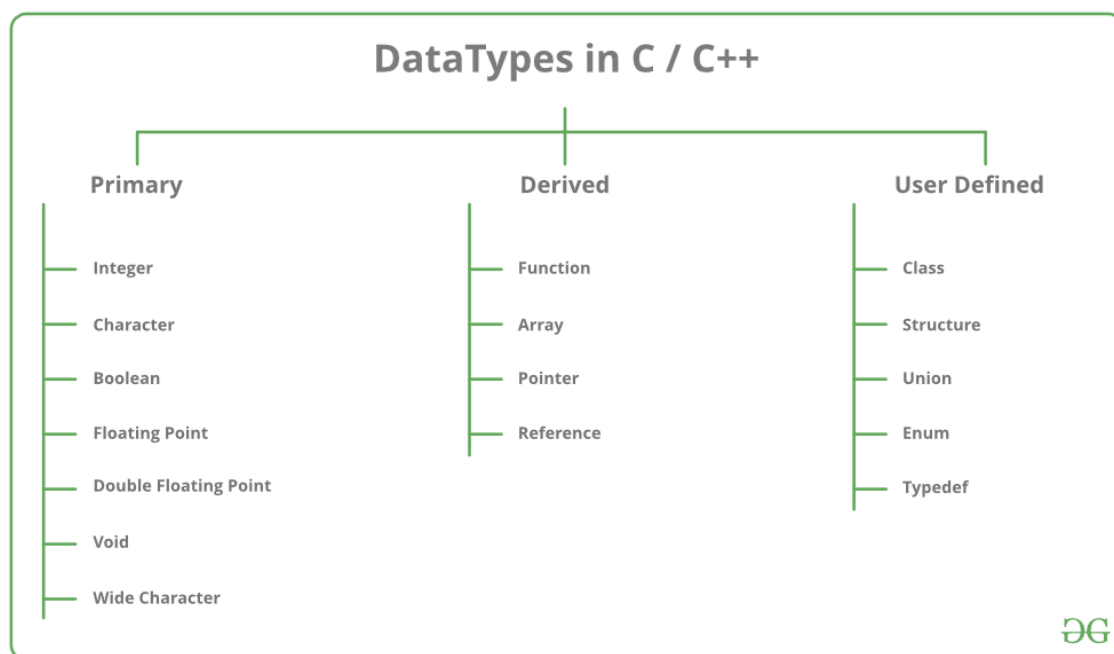


Figure 5 Types in C++

2.3.1 Primitive datatypes

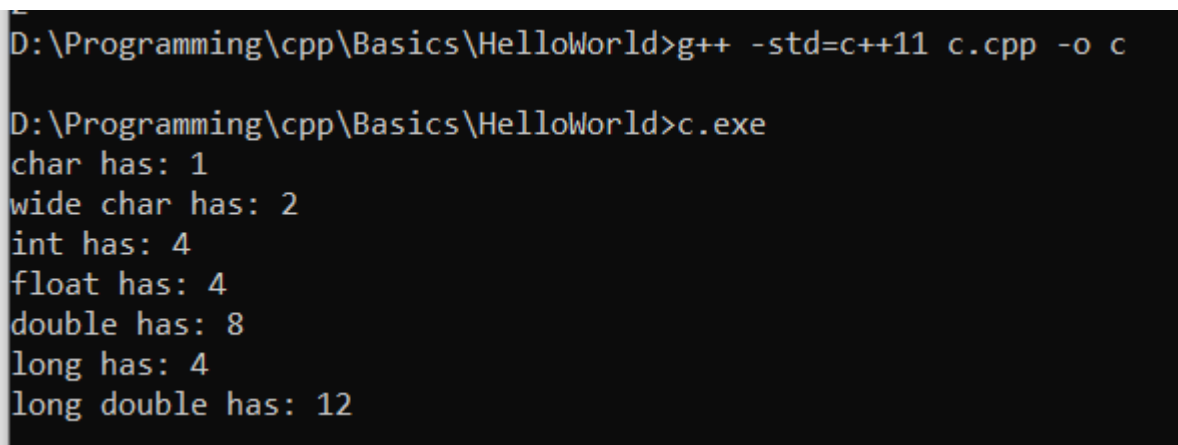
Primary (primitive) data types are compiler dependent that means that the data types could be stored in different sizes for different compilers, in gcc compiler:

Type the following to examine the sizes of different datatypes, for example int (integer saved in 4 bytes in gcc).

```
#include<iostream>
using namespace std;

int main() {
    cout<<"char has: "<<sizeof(char)<<endl;
    cout<<"wide char has: "<<sizeof(wchar_t)<<endl;
    cout<<"int has: "<<sizeof(int)<<endl;
    cout<<"float has: "<<sizeof(float)<<endl;
    cout<<"double has: "<<sizeof(double)<<endl;
    cout<<"long has: "<<sizeof(long)<<endl;
    cout<<"long double has: "<<sizeof(long double)<<endl;
    return 0;
}
```

The output should be in gcc compiler (maybe different for other compilers) see Figure 6:



```
D:\Programming\cpp\Basics\HelloWorld>g++ -std=c++11 c.cpp -o c
D:\Programming\cpp\Basics\HelloWorld>c.exe
char has: 1
wide char has: 2
int has: 4
float has: 4
double has: 8
long has: 4
long double has: 12
```

Figure 6 datatypes sizes in gcc compiler

WHY we use different types of primitive (primary) variables?

To answer this question let's examine the following table

	details	Memory allocation (in GCC)	Syntax
char	Store characters ('a','b',etc) and integers from -128 to 127	1	char x = 'a';
wchar_t	Store much more characters than char	2	wchar_t x = L'あ';
int	Store integer numbers till 2^{31} positive integers and 2^{31} negative integers	4	int x = 15;
float	Store decimal numbers	4	float x = 15.12;

Also you have some modifiers like long/short and signed and unsigned

- Short: shorten integer to be usually stored in 2 bytes instead of 4 bytes which means that the value of short int will from 2^{15} positives and 2^{15} negatives not 2^{31} positive integers and 2^{31} negative integers.
- Long: will long the integers to be usually 12 bytes instead of 4 bytes which enlarge the range of that variable
- unsigned: signed (char or int or even short int) will store all bytes in positive for example, unsigned char has range of 0-255 while signed char (or char) has -128 to 127 (2^7 positives and 2^7 negatives)

back to our question, why we have different primitive data types? simply if I have variable that store integer variable of human age, I want only a variable that store positive integers of range 0 yrs old -150 yrs old, so char will be chosen or even short int (aka short) no need to take 4 bytes of integer as no human ever lived 2billion years !! so it waste of memory to choose int.

remember ! char variable store integers like 15 and characters like 'a' not only characters

what happen if:

1. what happen if: signed short int (aka short) which have range of -32768 to 32767, store number like 32770?

ans: the variable will overflow (aka return to zero and start to count gain the reminder) which mean that 32770 is higher than the capability of unsigned short (32767) by 3 so the value will be 3 like in Figure 7 Variables overflow, note: same thing to unsigned short variable the start 0 and max is 65535 so if the number exceeds; it will start counting the reminder from 0.

Remember: when you exceed the variable range; overflow will happen

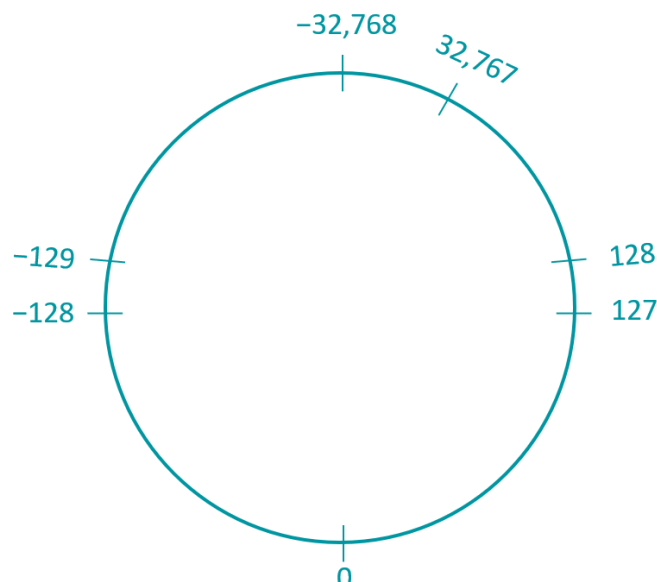


Figure 7 Variables overflow

2. what happen if: storing float number like 15.02 in integer variable like `int x = 15.02` ?

Ans: the float point (.02) will be truncated i.e. x is 15 only

SO always remember which primitive data types to choose !!;

Exercises on primitive (primary) Data types:

Exercises 1: introduction

Write C++ code to introduce someone, the introduction must include:

- Name (string): like “Ahmed” , to declare string datatype called string like:
string name;
cin>>name;
- Age (unsigned short) like 28
- Salary (unsigned short) like 15000
- GPA (float) like 3.5
- NOTE: the data should be as input from user: to get input from user use
cin>>var;

Answer:

```
#include <iostream>
using namespace std;

int main() {
    string name;
    unsigned short age,salary;
    float gpa;
    cout<<"enter your name"<<endl;
    cin>>name;
    cout<<"enter your age and salary "<<endl;
    cin>>age>>salary;
    cout<<"enter your gpa"<<endl;
    cin>>gpa;

    cout<<"Introduction\nMy name is:"<<name<<endl;
    cout<<"I am "<<age<<"years old "<<"my salary is: "<<salary<<endl;
    cout<<"my GPA is: "<<gpa;
    return 0;
}
```

NOTE: \n between “ ” is as same as endl after cout which means start from new line (i.e start printing at the beginning of the new following line)

NOTE: using namespace std; is used to write cout and cin without typing std::cout and std::cin

Exercise 2: bankClient

Write C++ program to show:

- Client name: string
- ID: int
- Deposit money: float

Answer in the GitHub repository: [LINK](#)

All the previous was all about primitive datatypes, but how about derived and user defined datatypes? Recall Figure 5 Types in C++

derived datatypes are datatypes made from primitive

- Arrays
- Functions
- Pointers

User defined datatypes are datatypes that user build

- Struct
- Enum
- Union
- Class

Lets take them one by one:

2.3.2 Derived datatypes

- **Arrays**

are list of some variables but must be same data type variable Like int list[3] clientAges; which means that we collect clientAges in one list instead of doing this: int client1Age; int client2Age; int client3Age;

So, to make the life easier we collect similar datatypes in one place called array

- **Declaration:** datatype nameOfArray[number of item];
For example: int salaries[5];
- **Accessing each element:** salaries[i] (i must be number from 0 to 4 as salaries have 5 items)

- **Functions**

Imagine you want to introduce 10 people (like in **Exercises 1**: introduction) the program was about 10 line for one person, do write same code for the 10 person (100 lines !!) OR you can write the code for general person once in a place called function and whenever you want to use that function, call that general function and specify your details

```
void introduction(string name, short age, short salary, float gpa ){
    cout<<"enter your name"<<endl;
    cout<<"enter your age and salary "<<endl;
    cout<<"enter your gpa"<<endl;
    cout<<"Introduction\nMy name is:"<<name<<endl;
    cout<<"I am "<<age<<"years old "<<"my salary is: "<<salary<<endl;
    cout<<"my GPA is: "<<gpa;
}
```

You build the general function, you can now call it as many times as you want !!

```
introduction("Ahmed",26,15000,3.6);
introduction(Gamal,30,2500,3.8);
introduction(Mahmoud,22,1200,3.2);
```

we will know more about functions and pointers later.

2.3.3 User-defined datatypes

- Structs

Struct is used when you want to declare an object that has many attributes (i.e. variable) but different data types, e.g. you want to describe a student who has name (String), id (int), gpa (float), struct came to hold these attributes (variables) in one place called struct

Example: studentStruct

In this example, struct is made for a student who has 3 attributes for example name (String), id (int), gpa (float).

```
//declaration
struct student{
    string name;
    int id;
    float gpa;
};

int main(){
    //create instance of a struct
    student Ahmed={"Ahmed",202410,3.45};
    /*Accessing
    Accessing is done by dot operator .
    */
    cout<<"Name:"<<Ahmed.name<<" ID:"<<Ahmed.id<<"
    GPA:"<<Ahmed.gpa<<endl;
    //Assigning an instance of struct
    Ahmed.gpa = 3.58;
    cout<<"Name:"<<Ahmed.name<<" ID:"<<Ahmed.id<<" GPA:"<<Ahmed.gpa;
}
```

NOTE: you can use comment to improve code readability:

- One line comment: using // comment
- Multiline comment: using /* comment */

1- Declaration of struct

```
struct name{
    variable1;
    variable2;
    .
    .

};
```

2- Creating instance

- 1st way: after the deceleration

```
//declaration
struct student{
    string name;
    int id;
    float gpa;
};
```

- 2nd way: by using.. struct_type struct_name;

```
student Ahmed={"Ahmed",202410,3.45};
```

NOTE: struct objects (instances) could be initialized of left to be assigned later

```
student Ahmed;
```

NOTE: in C++ you don't have to use struct keyword in contrast in C

In C:

```
struct student Ahmed={"Ahmed",202410,3.45};
```

in C++ struct is not necessary :

```
student Ahmed={"Ahmed",202410,3.45};
```

3- Accessing and Assigning

Accessing done by dot operator

e.g cout<<"Name:"<<Ahmed.name<<" ID:"<<Ahmed.id<<" GPA:"<<Ahmed.gpa<<endl;

Assigning:

```
Ahmed.name="Ahmed";
```

Exercise 3: employee

Write a struct that refer to an employee that have name , salary, working hours

The answer in basics folder in the repository, see Figure 8 Exercise 3

```
D:\Programming\MasteringCPP\Basics\VariablesAndDatatypes>employee.exe
enter Name, Salary, Working Hrs respctively:
Ahmed 15000 50
employee: Ahmed salary: 15000 working hours: 50
```

Figure 8 Exercise 3

4- Methods

Unlike C, in C++ we have methods in struct, methods are function inside structs or classes, Lets see how methods work

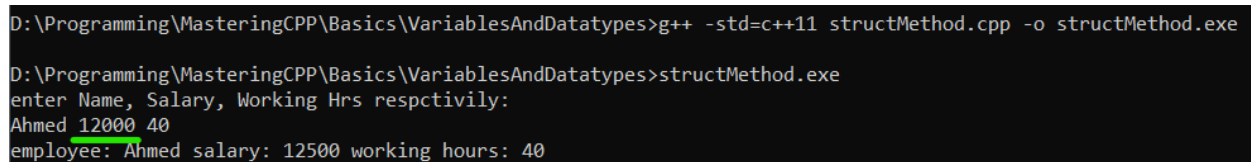
Example: structMethod

write employee struct that has name, salary, working hours, that get user data and print this data and apply bonus, so we must have 3 method(functions), see the output in Figure 9 Example

```
#include<iostream>
using namespace std;
struct employee{
    string Name;
    int salary;
    short workingHrs;

    //Method to enter employ data
    void setData() {
        cout<<"enter Name, Salary, Working Hrs respctively:\n";
        //entering the employee data from user
        cin>>Name>>salary>>workingHrs;
        //printing the employee data
    }
    //Method to print employee data
    void print() {
        cout<<"employee: "<<Name<<" salary: "<<salary<<" working hours: "<<work-
ingHrs<<endl;
    }
    //Method to apply bonus
    char applyBonus(int bonus){
        salary = salary + bonus;
        return 's';
    }
};

int main() {
    //create object of struct employee
    employee emp1;
    emp1.setData();
    emp1.applyBonus(500);
    emp1.print();
}
```



```
D:\Programming\MasteringCPP\Basics\VariablesAndDatatypes>g++ -std=c++11 structMethod.cpp -o structMethod.exe
D:\Programming\MasteringCPP\Basics\VariablesAndDatatypes>structMethod.exe
enter Name, Salary, Working Hrs respctively:
Ahmed 12000 40
employee: Ahmed salary: 12500 working hours: 40
```

Figure 9 Example

5- Constructors

Constructor is type of method that is called by default when an instance is made, the purpose of a constructor is to initialize the object, setting up initial values for its members and performing any setup required.

Example: structConstructor

```
#include <iostream>
using namespace std;

struct Person {
    string name;
    int age;

    // Constructor
    Person(string n, int a) : name(n), age(a) {
        cout << "Constructor called for " << name << endl;
    }

    // Member function to display person details
    void display() const {
        cout << "Name: " << name << ", Age: " << age << endl;
    }
};

int main() {
    // Creating an object of the Person struct
    Person person1("John Doe", 30);

    // Displaying the details of person1
    person1.display();

    return 0;
}
```

6- Access Modifiers : Public, Private, Protected