Modern C++

An effective short way

By

Mustapha Ossama Abdelhalim

2024

Contents

[1 Starter and Installation 3](#_Toc170604953)

[1.1 For windows 3](#_Toc170604954)

[1.2 For Linux 3](#_Toc170604955)

[2 Basics 5](#_Toc170604956)

[2.1 Introduction 6](#_Toc170604957)

[2.2 Hello World 7](#_Toc170604958)

[2.3 Variables and data types 8](#_Toc170604959)

[2.3.1 Primitive datatypes 9](#_Toc170604960)

[2.3.2 Derived datatypes 13](#_Toc170604961)

[2.3.3 User-defined datatypes 15](#_Toc170604962)

# Starter and Installation

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

## For windows

1. Go to [winlibs.com](https://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](https://www.youtube.com/watch?v=COZw6XetvR0)

## For Linux

Gcc is installed by default in ubuntu distribution

A screenshot of a computer

Description automatically generatedAfter 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 form environment variables if exists. See Figure 2 adding bin folder path to environment variables

Figure 2 adding bin folder path to environment variables

A screen shot of a computer error

Description automatically generatedType in cmd gcc –version and you should see that gcc installed see Figure 3 verifying gcc installation

Figure 3 verifying gcc installation

# 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

## 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

## 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 programA screenshot of a computer

Description automatically generated

Figure 4 first program

## 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, and120) or decimal aka float like (10.2, 0.2, and 22.8) or character (like ‘a’ , ‘b’ and ‘c’) , this declaration specify:

* + How the variable stored in memory and takes how much of program memory
  + How operation change that variable

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

Figure 5 Types in C++

### 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;**

**}**

A computer screen with white text

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

Figure 6 datatypes sizes in gcc compiler

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

To answer this question lets 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 32770is 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 65635 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

1. 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. s is 15 only

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

Exercises on primitive (primary) Data types:

**Exercises** : 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 namesapace std; is used to write cout and cin without typing std::cout and std::cin

**Exercise** : bankClient

Write C++ program to show:

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

Answer in the GitHub repository: [LINK](https://github.com/MuOssama/MasteringCPP/tree/main/Basics/VariablesAndDatatypes)

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:

### 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

The previous array called C-Array, C++ has much powerful arrays, these arrays have built-in method like size() and other to shorten your code

* + **Declaration**: array<datatype, itemNumbers> name;

For example: array<int, 5> salaries;

NOTE: don’t forget to include array (i.e #include <array>)

* + **Accessing each element**: salaries[i] (i must be number from 0 to 4 as salaries have 5 items

**Exercise** : arrays

Write C++ array of 5 integer contains some user salaries, don’t use c arrays, use C++ std array

#include<iostream>

#include<array>

**using** **namespace** std**;**

int main**(){**

array**<**int**,** 5**>**salaries**;**

//filling the array

**for(**int i**=**0**;**i**<**salaries**.**size**();**i**++){**

cout**<<**"enter the "**<<**i**<<**" element:"**;**

cin**>>**salaries**[**i**];**

cout**<<**"\n"**;**

**}**

//printing the array

**for(**int i**=**0**;**i**<**salaries**.**size**();**i**++){**

cout**<<**"the element "**<<**i**<<**" is: "**<<**salaries**[**i**]<<**"\n"**;**

**}**

**}**

* **Functions**

Imagine you want to intoduce 10 peaple (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.

### User-defined datatypes

* Structs

Struct is used when you want to declare and object that have many attributes (i.e. variable) but different data types, e.g you want to describe a student how have 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 student who have 3 atributes 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;

.

.

};

1. **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**};**

1. **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

A black background with white text

Description automatically generated

Figure 8 Exercise 3

1. **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 respctivily:\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: "**<<**workingHrs**<<**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**();**

**}**

A black screen with white text

Description automatically generated

Figure 9 Example

1. **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**;**

**}**

1. Inheritance

Inheritance used to create a child class of parent class or struct , e.g. if we created a class for employee that has name and age and member function named (method) role that is either writing() or reviewing() , we could create child of struct that inherit name and age but in writers employee child struct, writing() method will be created and in reviewer child struct, reviewing() method will be created.

**Example:** inheritance

#include <iostream>

#include <string>

// Base struct

struct Employee **{**

std**::**string name**;**

int age**;**

// Constructor for Employee

Employee**(**const std**::**string**&** n**,** int a**)** **:** name**(**n**),** age**(**a**)** **{}**

**};**

// Derived struct for Writer

struct Writer **:** public Employee **{**

// Constructor for Writer

Writer**(**const std**::**string**&** name**,** int age**)** **:** Employee**(**name**,** age**)** **{}**

// Specific method for Writer

void writing**()** const **{**

std**::**cout **<<** name **<<** " is writing a document." **<<** std**::**endl**;**

**}**

**};**

// Derived struct for Reviewer

struct Reviewer **:** public Employee **{**

// Constructor for Reviewer

Reviewer**(**const std**::**string**&** name**,** int age**)** **:** Employee**(**name**,** age**)** **{}**

// Specific method for Reviewer

void reviewing**()** const **{**

std**::**cout **<<** name **<<** " is reviewing a document." **<<** std**::**endl**;**

**}**

**};**

int main**()** **{**

// Create instances of Writer and Reviewer

Writer writer**(**"Alice"**,** 30**);**

Reviewer reviewer**(**"Bob"**,** 45**);**

// Use specific methods

writer**.**writing**();** // Output: Alice is writing a document.

reviewer**.**reviewing**();**// Output: Bob is reviewing a document.

**return** 0**;**

**}**

1. **Access Modifiers : Public, Private, Protected**

In the previous example, we could access display() method and any attribute (e.g name, age) anywhere, there are 3 places could a method or attribute called:

1. In the struct or class itself such enterData() call of age attribute check in the following example

struct Person **{**

string name**;**

int age**;**

// Member function to enter member data

void enterData**()** const **{**

cin **>>** name **>>** age**;**

**if(age<0)** cout **<<** “invalid age\n”;

**}**

// Member function to display person details

void display**()** const **{**

enterData();

cout **<<** "Name: " **<<** name **<<** ", Age: " **<<** age **<<** endl**;**

**}**

**};**

***All access modifiers are accessible within a class or struct***

1. In function like main() function after creating an instance of class of struct like person1.name = “void”, and person1.display();the following example:

int main**()** **{**

// Creating an object of the Person struct

Person person1**(**"John Doe"**,** 30**);**

// Displaying the details of person1

person1**.**name = “void”;

person1**.**display**();**

**return** 0**;**

**}**

***If age and name are private or protected, they wont be called outside the class or struct***

1. Last call or access of attributes and method (member function) is used in inheritance like public in line 12 the inheritance example:
2. // Base struct
3. struct Employee **{**
4. std**::**string name**;**
5. int age**;**
6. // Constructor for Employee
7. Employee**(**const std**::**string**&** n**,** int a**)** **:** name**(**n**),** age**(**a**)** **{}**
8. **};**
9. // Derived struct for Writer
10. struct Writer **:** public Employee **{**
11. // Constructor for Writer
12. Writer**(**const std**::**string**&** name**,** int age**)** **:** Employee**(**name**,** age**)** **{}**
13. // Specific method for Writer
14. void writing**()** const **{**
15. std**::**cout **<<** name **<<** " is writing a document." **<<** std**::**endl**;**
16. **}**

**};**

A screenshot of a computer

Description automatically generatedNote: the line struct Writer **:** public Employee is public inheritance see Figure 10 public, protected, private inheritance, members are attributes and methods

Figure 10 public, protected, private inheritance

The following table in Figure 10 Access Modifiers introduce how access modifiers work



Figure 11 Access Modifiers

For now we introduced only structs in user-defined data types, also we have union and enums

* Enum

Enum is abbreviation of enumeration, which used to give some related integers names as humans don’t remember and work with number well, e.g. if a worker get 500$ on Sunday and 600$ on Monday and 700$ on Tuesday ….. . an enum could hold these number and when we want give the worker 500$ on Monday, we could use Monday instead of using 500 number

**Example:** enum

Write C++ enum that define workday wage for a worker,

Sunday = 500, Monday = 600, Tuesday = 700, Wednesday = 800,

Thursday = 900, Friday = 1000, Saturday = 1100

#include<iostream>

**using** **namespace** std**;**

enum days**{**

Sunday **=** 500**,**

Monday **=** 600**,**

Tuesday **=** 700**,**

Wednesday **=** 800**,**

Thursday **=** 900**,**

Friday **=** 1000**,**

Saturday **=** 1100

**};**

int main**(){**

days workDay**;**

cout**<<**"Worker earned: "**<<**Sunday**<<**"$ wage"**<<**endl**;**

cout**<<**"Worker earned: "**<<**Monday**<<**"$ wage"**<<**endl**;**

cout**<<**"Worker earned: "**<<**Tuesday**<<**"$ wage"**<<**endl**;**

cout**<<**"Worker earned: "**<<**Wednesday**<<**"$ wage"**<<**endl**;**

cout**<<**"Worker earned: "**<<**Thursday**<<**"$ wage"**<<**endl**;**

cout**<<**"Worker earned: "**<<**Friday**<<**"$ wage"**<<**endl**;**

cout**<<**"Worker earned: "**<<**Saturday**<<**"$ wage"**<<**endl**;**

**}**

* Union

Union is user-defined data type that all attributes of that union share the same memory see Figure 12 Union vs struct, if I changed n in union; m will be changed too



Figure 12 Union vs struct

**Example:** union

Write C++ union that holds char x=1 and short y=65535 , show the size of the that union and change value of x to 2 and print y and values

#include<iostream>

**using** **namespace** std**;**

union storage**{**

unsigned char x**;**

unsigned short y**;**

**};**

int main**(){**

storage var**;**

var**.**x **=** 1**;**

var**.**y **=** 65535**;**

cout**<<**"size of var is: "**<<sizeof(**var**)<<**endl**;**

cout**<<**"x y resp: "**<<(**unsigned short**)**var**.**x**<<**" "**<<**var**.**y**<<**endl**;**

var**.**x **=** 2**;**

cout**<<**"x y resp: "**<<(**unsigned short**)**var**.**x**<<**" "**<<**var**.**y**<<**endl**;**

**}**

You can see the output in Figure 13 union example, x is unsigned char that holds 1 byte, while y is unsigned short that holds 2 bytes, the first byte is shared by x and y

Like in Figure 14 union example explanation

A black screen with white text

Description automatically generated

Figure 13 union example

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
|  |  |  |  |  |  |  |  | X = 255 | | | | | | | |
| Y = 65535 | | | | | | | | | | | | | | | |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  | when x changed to 2, y is affected as they have 1 byte shared | | | | | | | | |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 2nd byte | | | | | | | | 1st byte | | | | | | | |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
|  |  |  |  |  |  |  |  | X = 2 | | | | | | | |
| Y = 65282 | | | | | | | | | | | | | | | |

Figure 14 union example explanation

**Bitfield**

Bitfield is used in struct and union to specify bit values, e.g. if we have an 8bit register that we want to change every bit, we could do that.

**Example:** bitfield

Write a bitfield to mimic an 8bit register by union

#include<stdio.h>

**using** **namespace** std**;**

union Reg**{**

struct**{**

unsigned char B0**:**1**;**

unsigned char B1**:**1**;**

unsigned char B2**:**1**;**

unsigned char B3**:**1**;**

unsigned char B4**:**1**;**

unsigned char B5**:**1**;**

unsigned char B6**:**1**;**

unsigned char B7**:**1**;**

**}**Bits**;**

unsigned char byte**;**

**};**

int main**(){**

Reg DDRA**;**

DDRA**.**Bits**.**B0**=**1**;**

DDRA**.**Bits**.**B1**=**1**;**

DDRA**.**Bits**.**B2**=**1**;**

DDRA**.**Bits**.**B3**=**0**;**

DDRA**.**Bits**.**B4**=**0**;**

DDRA**.**Bits**.**B5**=**0**;**

DDRA**.**Bits**.**B6**=**0**;**

DDRA**.**Bits**.**B7**=**0**;**

printf**(**"%d"**,**DDRA**.**byte**);**

**}**

NOTE: in this example, printf must be used instead of cout, so we have to include stdio.h library