**C++**

* Bjarne Stroutstrup is the creator and developer of C++ Programming Language in 1979.
* It was initially called as “C with Classes” until 1983
* C had its limitations.
  + Computational Capacities of the Computers and the Programming complexity increased year-by-year
* Object Oriented Programming (OOP) was introduced in C which was based on Simula 67
* General Purpose Programming Language – used for creating a variety of applications
* Compiled Language
* C++ is a General Purpose Middle Level Compiled Object Oriented Programming Language
* Code written in any Programming Language is called the Source Code
* File containing the Source Code is called Source File
  + C++ File 🡪.cpp, .cc, .cxx, .cp, .C (contains implementation)
  + Header File 🡪 .h, .hpp (contains declaration of features)
* C++ Standard Library has all the built-in features
* External Library has features included by other programmers
  + Static External Libraries
  + Dynamic External Libraries
* We need to include the header files for the Libraries
* https://isocpp.org

# Types of Languages

* Lower Level Language
  + Machine Language
  + Assembly Language
* Middle Level Language
  + C++
* Higher Level Language
  + Human Readable format
  + Built-in Features

# Types of Programming in C++

* Procedural Programming
* OOP
* Generic Programming (using Templates)
* Functional Programming (new in C++)

# Advantages

* Efficient Language
* Reliable and Fast
* Better understanding of OOP
  + Classes
  + Objects
  + Inheritance
  + Polymorphism
* Learning other languages will be easy since (eg. Java and C# since they also use OOP)

# Uses

* Games
* GUI Applications
* Web Browsers
  + Mozilla Firefox
* Compliers
* Operating Systems
* Photoshop
* Medical and Engineering Applications
* Creating other Programing Languages
  + Java

# Build Process

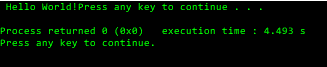
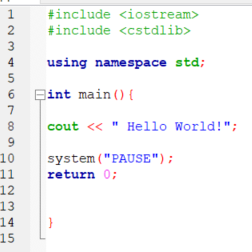
* The process of creating an executable file from the Source Code is called Build Process.
* The Step in a Build Processor in order to create an executable file is called Pre-processing.
  + Pre-Processor is a macro-processing program.
  + It takes the C++ Source File as the input.
  + It cleans up your code (removes spaces and comments)
  + It will contain the contents of the header files
  + It generates an intermediate Code
  + Output will be .ii extention
* Next Step is to translate the .ii files into machine code language for which we use the Compiler. Compiler is a computer program which transforms the code from Higher Level Programming Language to Lower Level Programming Language. This process is called Compilation. Compiler will also make sure that we have followed all the rules of C++.
  + All lines should end with ;
  + GNU, Turbo, Visual
* The Assembly Code is generated and the File generated is called Assembly File with .s extension
* To convert the Assembly Code to Machine code we use the Assembler which will generate the Object code/Machine Code. The File is called Object Code with .obj or .o extension
* Then the Linker will take the Object Files generated, link them together and generate the executable files. The process is called the Linking Process.
  + If the libraries are not found, it will generate an error.
* Loader will then load the Executable file into the Main Memory (RAM) so that the CPU can execute the Program to get the Output.

# NameSpace

* NameSpace is a declarative region in which various program elements can be placed.

# Main Function

* The point where the execution of the program starts.
* A Function is a collection of Statements which are going to do a particular task. They contain the actions that we are to perform.
* ‘cout’ stands for ConsoleOutput to print
* ‘<<’ is called Stream Insertion Operator
* ‘return 0;’ indicates that the program has successfully executed.
* Whatever we write in between the { } of a function definition is called a Function Body



# Constants

* A Constant is an entity which doesn’t change
  + Integer Constant Constants (44, 86)
  + Floating Point Constants (12.23, 34.53)
  + Character Constants (‘a’, ‘b’, ‘z’)
  + String Constants (“Dhruven”, “Lad”)

# Variables

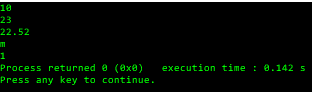
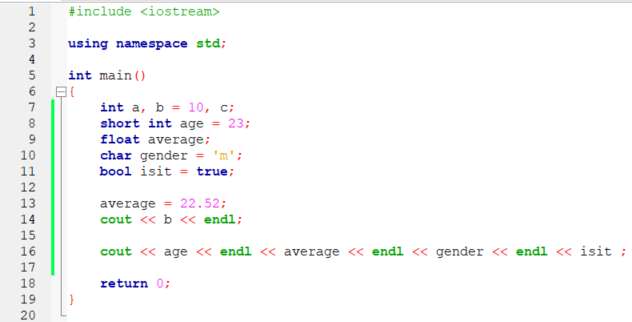
* A Variable is a name given to a memory location in order to access it easily

# DataTypes

* The type of data stored in a variable
  + Integer (4 bytes)
  + Double (8 bytes)
  + Character
  + Floating Point
  + Boolean (1 byte)
  + Void (used with functions)
* Signed, unsigned, short and long TypeModifiers used with the DataTypes. They influence the number of bytes to store a particular value
* Cannot start with special characters or numbers
* The only special character that can be used is the underscore (\_), and is usually used to separate words if camel casing is not used
  + Emp\_first\_name or empFirstName
* Variables start with lowercase letters, and names are indicative to the use of the variable

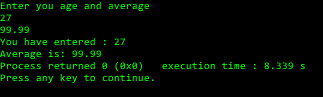
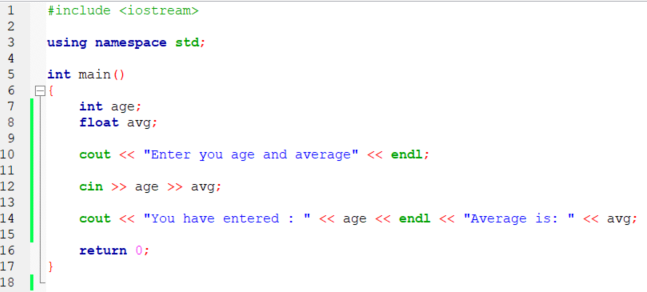
# KeyWords

* int, float, char, bool, if ,for, switch which cannot be used as variable names
* ‘endl’ to end line
* True = 1, False = 0



# CIn

* Used to read the Inputs from the KeyBoard entered by the User
* ‘>>’ is called Stream Extraction Operator

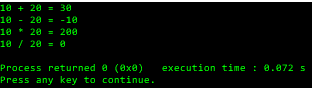
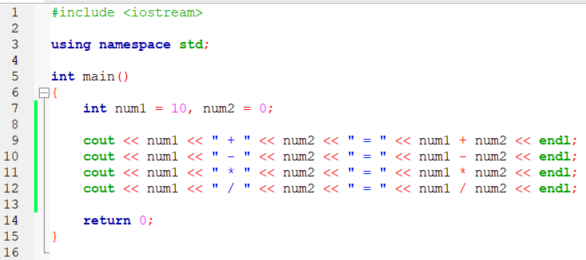


# Comments

* Comments are the descriptions written in Human readable format explaining what a particular block/statement does.
* Two types of Comments:
  + Single-line Comments ( // )
  + Paired Comments( /\* \*/ )

# Arithmatic Operators

* +, -, \*, /, %, +=, -=, \*=, /=, %=
* num1 += num2 🡪 num1 = num1 + num2;

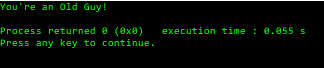
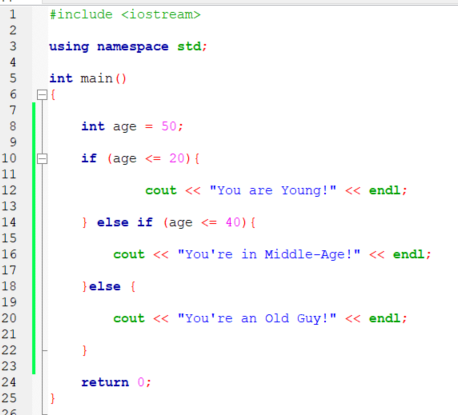


# Increment and Decrement Operators

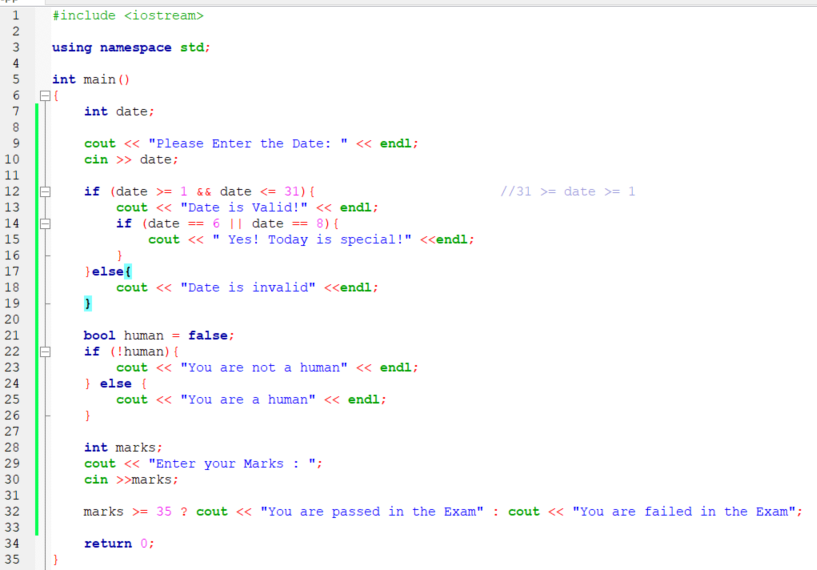
* ++a or a++, --a or a--
* Prefix or Postfix

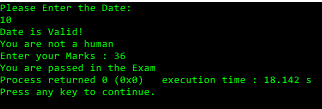
# If-Else

* If(condition){  
   ------  
   ------  
   }
* Conditions are the Comparison Operators
  + ==
  + >=
  + <=
  + !=



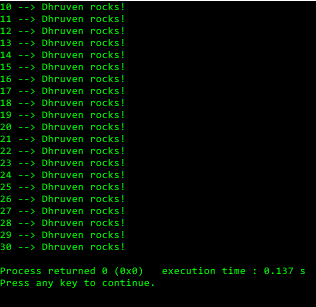
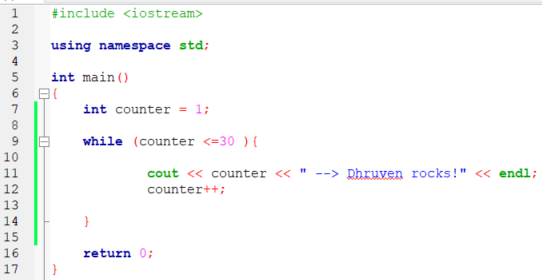
* Logical operators
  + &&
  + ||
  + !
* Ternery Operators
  + Expression ? true statement : false statement;





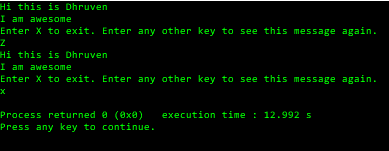
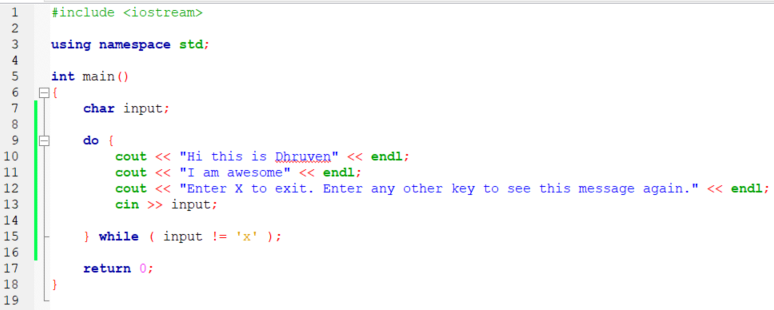
# While Loops

* While (expression){  
   statements  
  }
* Change the value of counter inside the loop



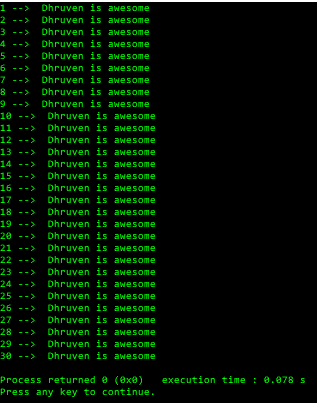
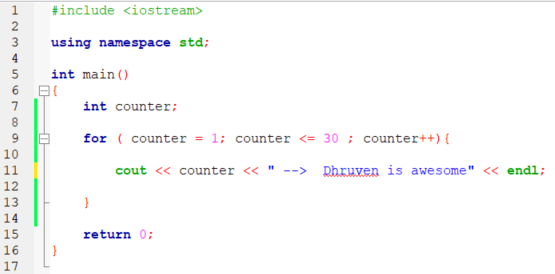
# doWhile Loops

* do {  
   statements  
  } while (expression);
* For Making Menus
* To execute the condition at least ONE time



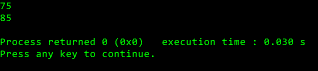
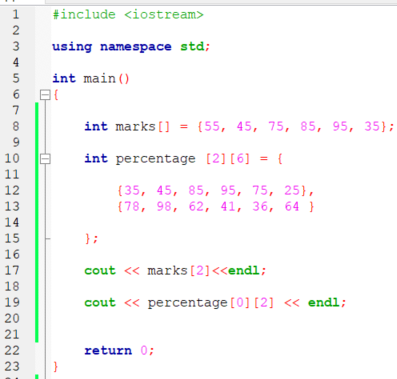
# For Loops

* for( init ; condition ; updation ){  
   statement  
   }
* Infinite Loop 🡪   
  for ( ; ; ){  
   }



# Arrays

* Collection of Elements.
* All elements must be of same type
  + Int
  + Char
* The values of the Arrays will be stored in continuous memory locations.
* Index from 0 to (n-1)
* A Multi-dimensional Array is an Array inside another Array
  + Marks [x][y], x= rows, y=column

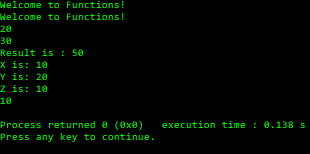
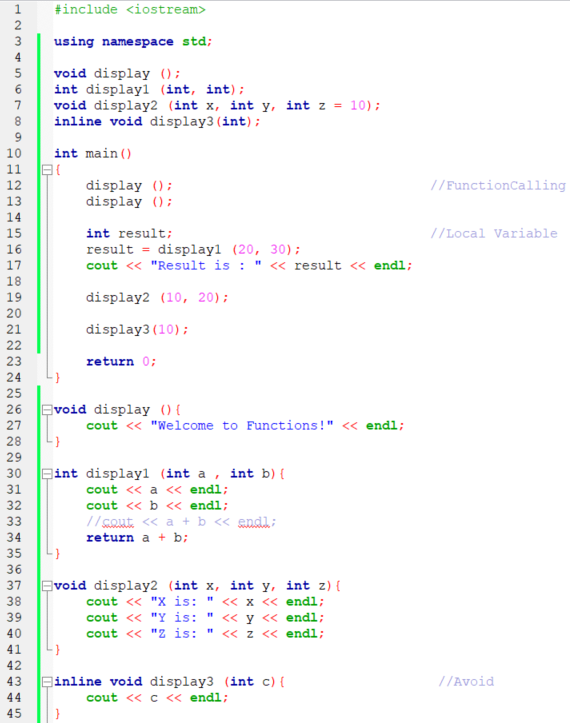


# Functions

* Depending on what the function is returning, we define the function data-type. (Return-type)
  + Int
  + Char
  + Void
* The Function needs to be called to be executed
* Function Prototyping is declaring just the Function (not the Function Body)
* String 🡪 int string.length()
* Iomanip 🡪 setw(), setfill(), setprecision()
* Cctype 🡪 bool isdigit(char), isalpha(char), isupper(char), islower(char)
* Cstdlib 🡪 rand(), srand(int)
* Name 🡪 FuntionName
* <Return Data Type> FunctionName (parameters list){Function Body Code}
* Defined at the end so that you don’t have to scroll through them all to get to the main code
* Prototyping is <Return Data Type> FunctionName above the main()

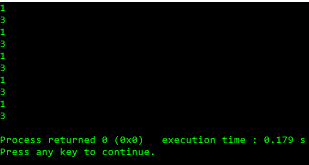
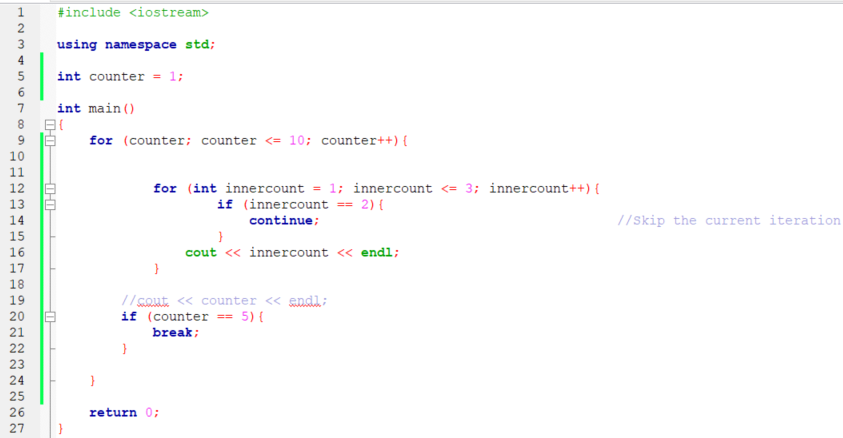
# Scope

* Scope is the region of a Program
* Variable Scope the region of a Program where a variable is accessible in a program
  + Local Scope (defaults: Garbage Value)
    - Inside a Function
    - While defining function (Formal Parameters)
  + Global Scope (defaults: int = 0, char = “ ”, pointer = NULL)
    - Outside function
  + Block Scope
* Block is the region which contains the code {…}



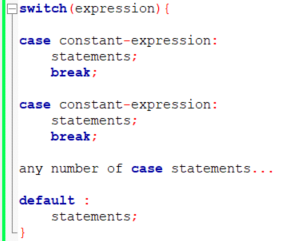
# Break

* The Loop immediately get terminated
* Terminate the Case in Switch Statements

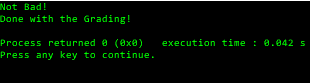
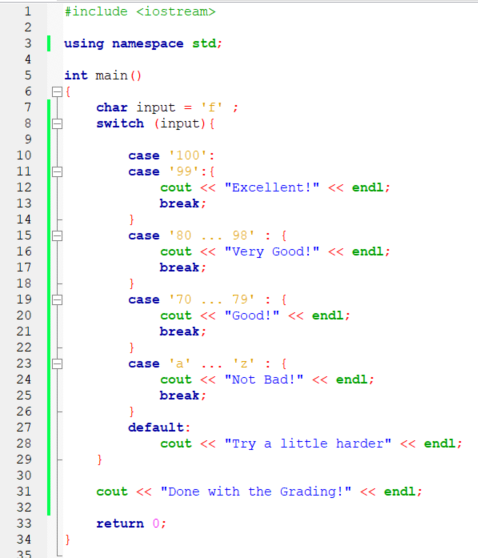


# Switch

* The Data-Types must be constant throughout
* We can have only one variable

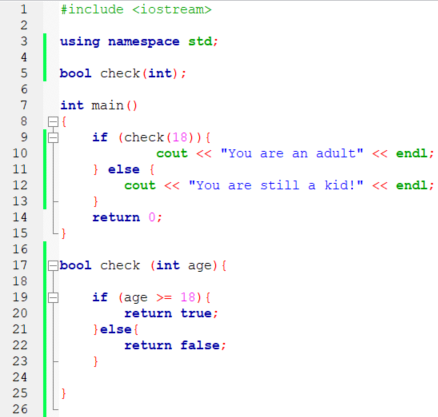


* Range 🡪 case low … high :
* No overlapping



# Multiple Return Statements

* We can use multiple return statements in a function but only one will be executed

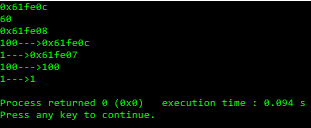
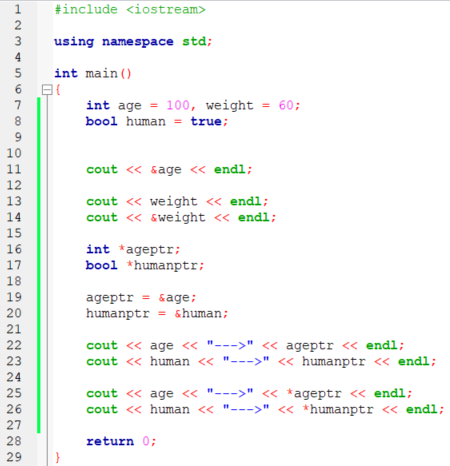


# Address Operator

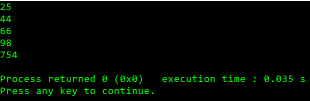
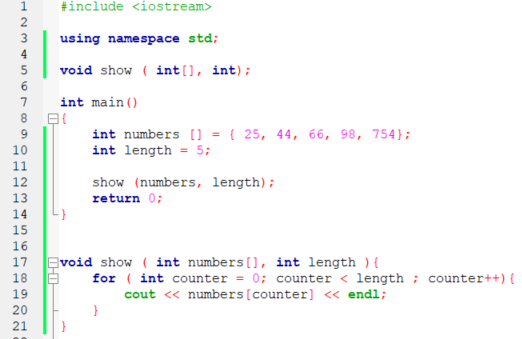
* & is used to get the address of the memory location of the element

# Pointers

* Pointer is a variable which contains the address of the memory location
* Dynamic Memory Operators need Pointers
* \* 🡪 Value At Operator

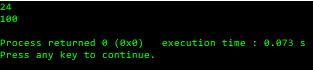
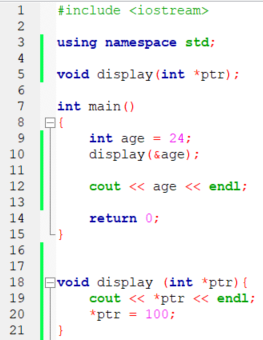


# Array to Function

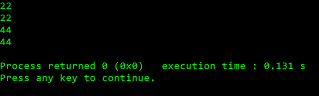
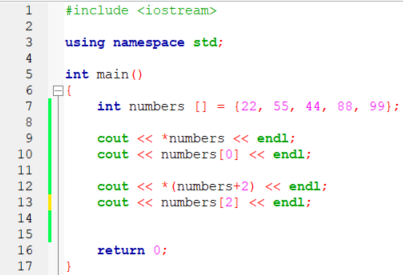


# Pointer to Function

* When we pass the values of a variable it is called Pass by Value and when we pass the address by Pointer to a function it is called Pass by Reference

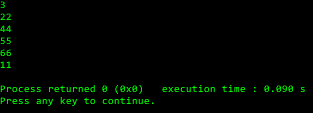
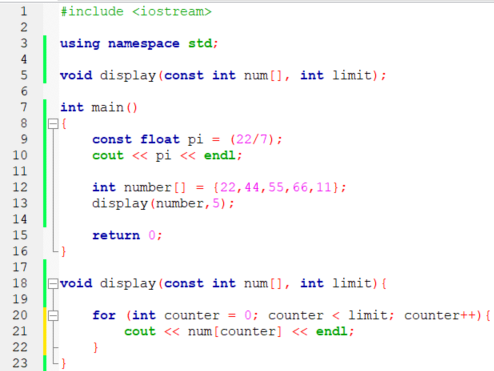


# Arrays and Pointers

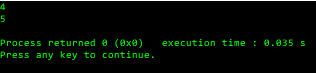
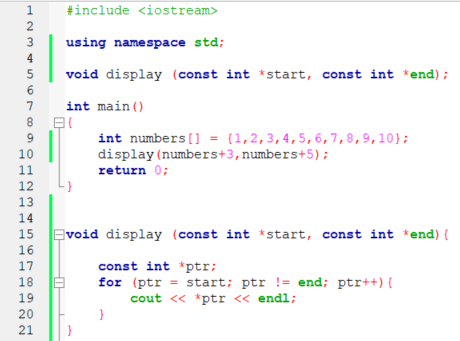


# Const

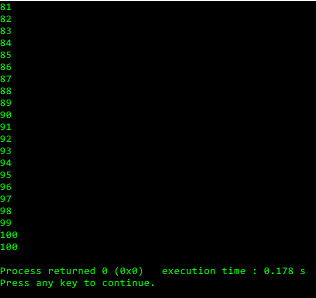
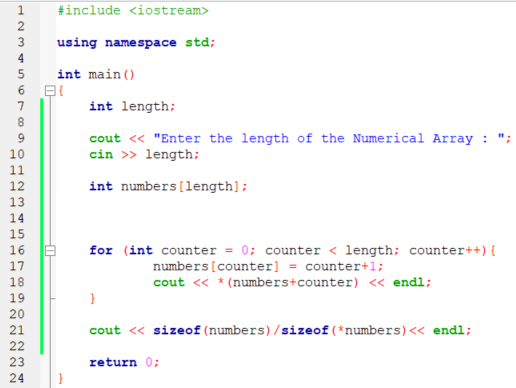
* Const is used to make the elements of the programs like variables and pointers as constants



# Array Ranges

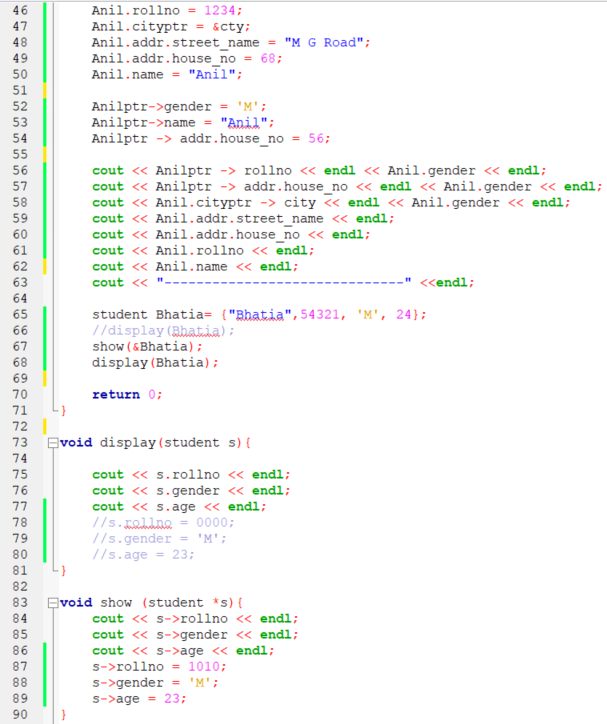
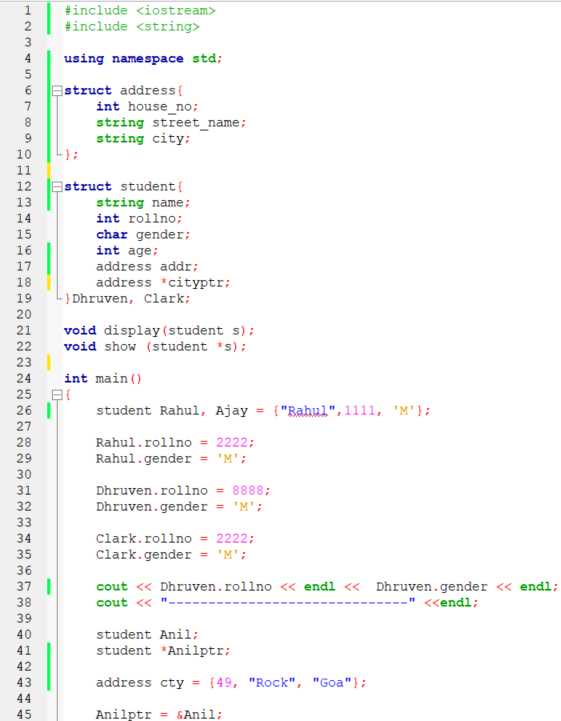


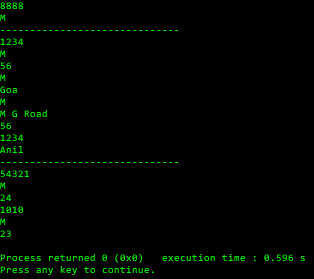
# Array Length



# Structures

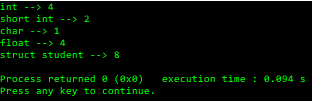
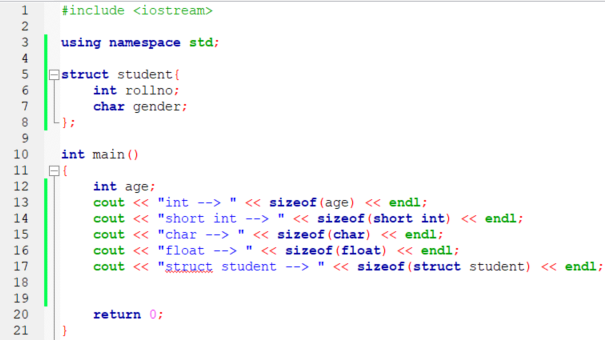
* Structures are the collections of Variables of different types
* {members of the structure - variables}
* For Variables we use Dot(.) operator
* For Pointers we use Arrow(->) operator





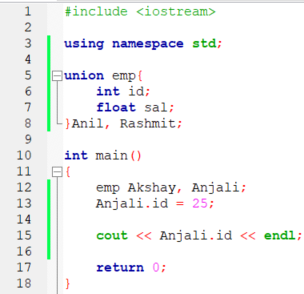
# Sizeof

* Used to determine the size of a variable of any data-type in bytes
  + Int
  + Float
  + Char
  + Unions
  + Classes



# Union

* Union is a User Defined Data-type
* Only one member at a time unlike Structures since the memory is shared here compared to structures where each member is allocated different memory



# Dynamic Memory Allocation

* new datatype; 🡪 For allocating the memory to heap dynamically and will return the address of the memory
* delete datatype; 🡪 To deallocate the memory

Text

Description automatically generated Graphical user interface, text

Description automatically generated

# User Input Arrays

Text

Description automatically generated Text

Description automatically generated

# Dangling Pointers

* Dangling pointer is a reference to a memory location which was previously used by the pointer but currently it is deallocated.
* Delete the pointer in the end and add null pointer later too.

Text

Description automatically generated

# Automatic Type Deduction

* Initialization is a must.



# Range-based For Loop

* For (variable : range) {  
  }

Text

Description automatically generated Text

Description automatically generated

# Strings

* #include <string>
* #include <cstring> 🡪 for C Language Strings

Text

Description automatically generated Graphical user interface, text

Description automatically generated

# Recursive Functions

* A function calling itself.
* There must be a point where the call stops.

Text

Description automatically generated 

# Function Overloading

* Functions with the same name and return type but different parameters are said to be overloaded functions.
* The return type must be same

Text

Description automatically generated Graphical user interface, text

Description automatically generated

# Object-Oriented Programming

* When we use functions for the programming, it is called structed programming, whereas when we use classes, it is called Object-Oriented Programming.
* A class is a user-defined data-type which allows us to group the data, activities and the actions together.
* The variables and functions defined within the class are called the Members of that class.
* Access Specifiers inform us where the classes are going to be used.
  + public 🡪 can be used outside the class
  + private 🡪 can be used only inside the class
  + protected
* Object is the variable of a class.
* To use the members of a class, we need to use the Dot Operator (.)

Text

Description automatically generated Text

Description automatically generated

Text

Description automatically generated Text

Description automatically generated

* Class Object 🡪 in this way, the object will be stored in the stack, not in the heap.
* Class \*Object = new Class () 🡪 Create object dynamically and stored in heap and since ‘new’ uses address, we need \*.

Text

Description automatically generated

# Scope Resolution Operator

* We can define the method of the class outside the class using a scope resolution operator ( :: )

Text

Description automatically generated Text

Description automatically generated

# Private Access Specifier

# Rand()

* Will generate a random number between 0 and 1.

Text

Description automatically generated 

# File I/O

* #include <fstream>
* .length

https://learn.epam.com/detailsPage?id=39063a5a-d6e9-4df8-9767-e3528d4c5ef7