3.3

C++ Program Structure

Source file Name – “helloWorld.cpp”

```
#include <iostream>  // header file { for multiple line comment /* header file */
using namespace std;  // Namespace
Return_type // int main()  //main method
        {
                cout << “Hello World\n”;  // print statement
                return 0;
        }
```

- The preprocessor is a program that processes the source code before it passes through the compiler.
- The preprocessor directive #include tells the compiler to insert another file into your source file.
- Eg: #include<iostream.h>
- #define A 20

Namespace

- a **namespace** is a collection of related names or identifiers (functions, class, variables) which helps to separate these identifiers from similar identifiers in other namespaces or the global namespace.
- the identifiers of the C++ standard library are defined in a namespace called std.
- Eg:

```
        #include<iostream.h>
        int main() {
                int n;
                std:: cout<<“enter any number”;
                std:: cin>>n;
                 std:: cout<<“the entered number is”<<n;
    return 0;
    }
```

- We used std::cin and std::cout instead of simply cin or cout. The prefix std:: indicates that the names cout and cin are defined inside the namespace named std.
- ::-> scope resolution operator

Input/Output Stream (cin and cout)

- Cin -> input
- Syntax: cin>>variables; // >> -> extract operator, cin -> get values from user.
- Cin is used for accepting data from keyboard
- >> (bitwise right shift operator) can be overloaded
- Eg: accepting input from user:

```
#include<iostream.h>
Using namespace std;
Int main(){
        int n;
        cout<<“enter number:”;
        cin>>n;
        cout<<“the entered number is:”<<n;
return 0;
}
```

- We can cascade multiple variables in cin.

```
Int  a,b;
Cin>>a>>b; or cin>>a; cin>>b;
```

- Extractor operator is also used to getting string. Eg: cin>>string;
- Cin allows to enter one word at a time. For reading entire line we used getlin();

cout

- Cout -> output
- Syntax: cout<<variable; // << ->insertion operator, cout-> display output to user using screen.
- << -> (bitwise left shift operator) can be overloaded.
- Tells the user to do appropriate action.
- We can cascading multiple variables in cout.

```
Int a=10;
Int b=20;
Cout<<a<<b; or cout<<a; cout<<b;
```

Use of expression in cout

```
int n1;
int n2;
cin>>n1;
cin>>n2;
cout<<"addition:"<<n1+n2;
```

Display output on new line: ‘n’ or ‘endl’

```
Cout<<"hello,n";
Cout<<"my name is  n Patan";
Cout<<"hello"<<endl;
Cout<<"my name is "<<endl;
Cout<<"Patan"<<endl;
```

Function Overloading:

- Overloading means the use of the same thing for different purposes.
- One function name but with different argument lists.
- The function would perform different operations depending on the argument list in the function call.
- The correct function to be invoked is determined by checking the number and type of the arguments but not on the function type.
- Declaration:

```
int add(int a, int b);
int add(int a, int b, int c);
double add(double x, double y);
double add(int p, double q)
 double add(double p, int q);
```

- Function call:

```
Cout<<add(5,10);
Cout<<add(15,10.0);
Cout <<add(12.5,7.5);
Cout<<add(5,10,15);
Cout<<add(0.75,5);
```

- The function call first matches the prototype having the same number and type of arguments and then call appropriate function for execution.

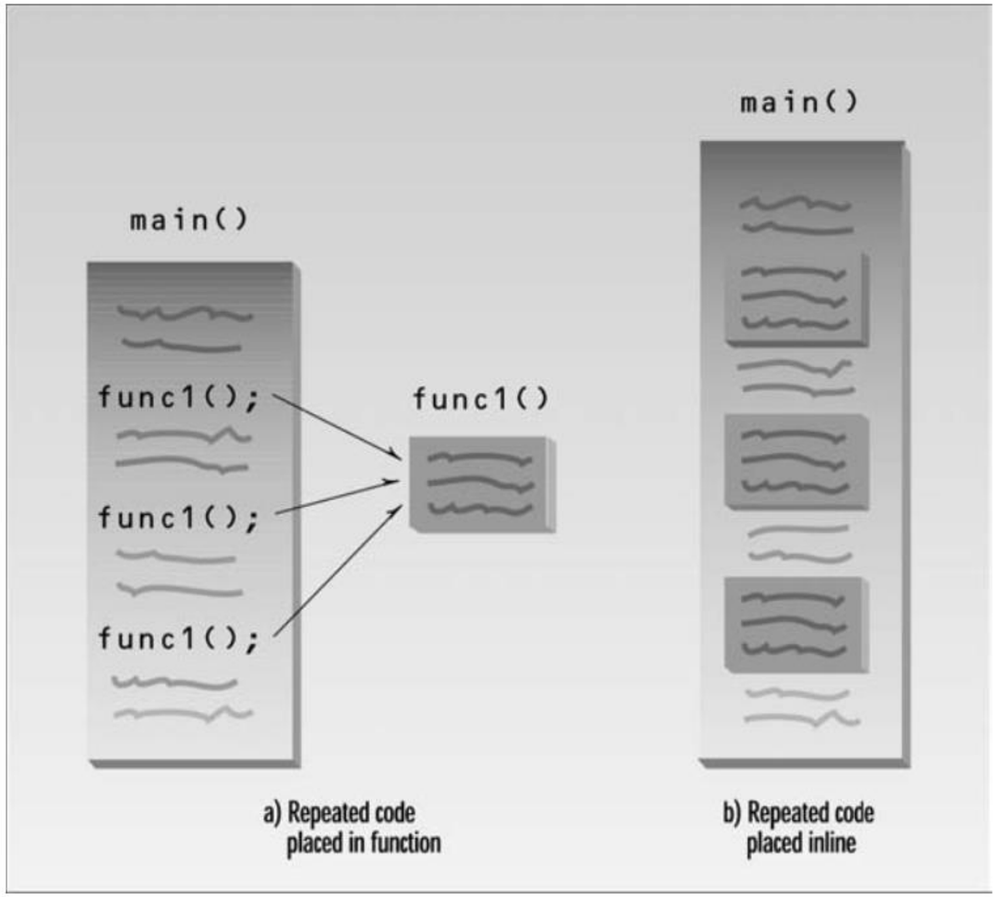
```cpp
#include <iostream>
using namespace std;
void SumNum(int A, int B);
void SumNum(int A, int B, int C);
void SumNum(int A, int B, int C, int D);
int main() {
SumNum(1,2);
SumNum(1,2,3);
SumNum(1,2,3,4);
return 0;
}
void SumNum(int A, int B) {
cout<< endl << "SUMNUM is : "<< A+B;
}
void SumNum(int A, int B, int C) {
cout<< endl << "SUMNUM is : "<< A+B+C; }
void SumNum(int A, int B, int C, int D) {
cout<< endl << "SUMNUM is : "<< A+B+C+D;
}
```

Inline Function:

- The inline keyword is actually just a *request* to the compiler to substitute the code within the function definition for every instance of a function call.
- Syntax:

```
inline return_type function_name(data_type arg1 ,
data_type arg2 )
              {
                // definition of the function
              }
```

- Reduce the line of code.
- Avoid again and again function calls.
- Memory space and time wastage reduce.


main()
func1();
func1();
func1();
func1()
main()
a) Repeated code placed in function
b) Repeated code placed inline

Eg: Calculate Area

```
Eg: #include<iostream>
using namespace std;
                 inline int area(int length, int breadth) { // define an inline function to calculate the area of a rectangle
                   return length * breadth ; }
                 int main() {
                 int length, breadth, result;
                  cout <<  "Enter the length of the rectangle: ";
                 cin >> length;
                  cout <<  "\nEnter the breadth of the rectangle: ";
                 cin >> breadth;
                     result = area(length , breadth);  // call the inline function (i.e. area())
// this will be replaced by the // definition of the inline function
                     cout <<  "\nThe area of the rectangle is: " << result;
                   cout << "\n\n";
                  return 0 ;
}
```

Defining inline function outside the class

```
Class item
{
Public:
Inline Void getdata(int a , float b);
};
Inline void item :: getdata(int a , float b)
{
Number= a;
Cost=b;}
```

Default Arguments

```
// demonstrates missing and default arguments
#include <iostream>
using namespace std;
void repchar(char='*', int=45); //declaration with default arguments
int main()  {
repchar(); //prints 45 asterisks
repchar('='); //prints 45 equal signs
repchar('+', 30); //prints 30 plus signs
return 0; }
// repchar()
// displays line of characters
void repchar(char ch, int n)   {
 for(int j=0; j<n; j++)
      cout << ch;
cout << endl; }
```

- In this program the function repchar() takes two arguments. It's called three times from main(). The first time it's called with no arguments, the second time with one, and the third time with two. Why do the first two calls work? Because the called function provides default arguments

Pass By Reference:

```
#include <iostream>
using namespace std;
void swapNums(int &x, int &y) {
  int z = x;
  x = y;
  y = z;
}
int main() {
  int firstNum = 10;
  int secondNum = 20;
 cout << "Before swap: " << "\n";
  cout << firstNum <<endl << secondNum << "\n";
  swapNums(firstNum, secondNum);
 cout << "After swap: " << "\n";
  cout << firstNum <<endl<< secondNum << "\n";
  return 0;
}
```

- to pass the reference of an argument in the calling function to the corresponding formal parameter of the called function. The called function can modify the value of the argument by using its reference passed in.

Return by Reference:

- In C++, reference can also be returned from a function.
- Eg: int &func(){

  return a; }

- Rules:
  - Returned variable must be a global variable.
  - Can not return constant from function.

Eg:

```
#include <iostream>
Using namespace std;
Int a;
Int &func() {
                return a;}
Int main() {
Func()=10;
Cout<<a;
return 0;
}
Output:10
```

Class:

- Class is used to hold data and functions together whereas structure only for data.
- By default the member of a class is private and the member of a structure is public.
- Syntax: class class_name;

```
                {
                        private:
                        variable_declarations;   } Class members
                        function_declarations;
                        public:
                        variable_declarations;   } Class members
                        function_declarations;
                };
```

-eg: Class student

```
        {
        int roll;
        char name[5];
        get_data();
        display_data();
        };
```

Access specifier/class member visibility

- It is used to control access to member within program.
- Three types: private (default)

  public

  protected (inheritance)

- Class members declared as private can be accessed only from within class.
- Public members can be accessed from outside the class also.
- The variable declared inside class are known as data members and functions are known as member function.
- Only member functions can have access to the private data members and private functions.
- The binding of data and functions together into a single class is known as encapsulation.

Eg:

```
Class student
{
        private:
        int roll;
        char name[50];
        float marks;
        public:
        float cost;
        void getdata(); // for input data
        void putinfo(); // for output data
};
```

public	The code is accessible for all classes
private	The code is only accessible within the same class
protected	The code is accessible within the same class, or in a class that is inherited from that class. You will learn more about inheritance in a later chapter

Creating objects:

- Once a class has been declared, we can create variables of that type using the class name.
- Eg: item x; (memory for x is created).
- Class variables are known as object.
- So x is object of type item.
- Objects can be also created when a class is defined by replacing their names immediately after the closing bracket.

```
Eg: class item
        {
                -----
                ------
        } x,y,z;// object of type item.
```

Object accessing class members:

- Syntax: object_name.function_name(arguments);
- Eg: x.getdata(100,5);

  x.putinfo();

- A member function is invoked only by using an object.
- Object can not directly access private data members.
- But publicly declared data and functions can be directly accessed by object.

Defining member function:

- Two places:

1. Outside the class definition
2. Inside the class definition
3. Outside the class definition:

- Member functions that are defined inside a class have to be defined separately outside the class.
- Syntax: return_type class_name :: function_name(arguments)

```
{
//function body
}
```

- the membership label “class_name ::” tells the compiler that the function belongs to the class.
- Eg: class item

```
        {
                int no;
                float cost;
        public:
                void getdata(int a, float b);
                void putdata(void);
        };
    Void item :: getdata(int a, float b){
        no =a;
        Cost=b;
    }
```

-different class can use the same function name with the help of :: scope is determined.
-Member functions can access the private data of a class.
-Non-member function can not do so.(friend function)
-A member function can call another function directly, without using dot operator.

Inside the class defination:

- Another method of defining a member function is to replace the function declaration by actual function definition inside the class.

Eg: class item

```
{
        int num;
        float cost;
        public:
        void getdata(int a, float b);
        void putdata(void)      //definition inside the class
        {        cout<<num<<endl;
                 cout<<cost<<endl;
        }
```

- When function is defined inside a class it is treated as an inline function.

Constructor and destructor:

- Initialization of objects when they are created and destroy them when their presence is not required.
- C++ provides a special member function constructor that enables an object to initialize itself when created.
- This is known as automatic initialization of objects.
- Another member function destructor destroys the object when not required anymore.
- **Constructor:**

```
Syntax:      class class_name
                  {
                  ----------
                  public:
                  ------------
                  class_name() //constructor declared should be in public section
                  {
                  ---------- // constructor definition inside class
                  }
             };
Class_name :: class_name()
{
------------// constructor definition outside the class
}
```

-can not returns value.

Eg:

```
class integer
{
        int m,n;
        public:
        integer(); // constructor declared
};
Integer :: integer() // constructor defined
{
        m=3;
        n=4;
}
```

- Constructor is a special member function that initializes the objects of its class.

-Its name is the same as the class name.

-Constructor is invoked whenever an object of its associated class is created.

Three types of constructors.

1. Default Constructor
2. Parameterized constructor
3. Copy constructor

1.**Default Constructor:** Default constructor is the constructor which doesn’t take any argument. It has no parameters. It is also called a zero-argument constructor.

Eg:

```
#include<iostream>
using namespace std;
class cube      {
                public:
                int side;
                cube()
                {
                side= 10;
                }
};
int main() {
                cube c;
                cout<<c.side;
                return 0;
}
Output : 10
```

Friend function:

- We know that, Private members can not be accessed outside the class and a non member function can not have an access to the private data of a class.
- There is a solution, where 2 classes can share a particular function.
- A common function can be made friendly with multiple classes and that function is allowed to access private data.
- That function need not to be a member of any of these classes.
- Syntax: class class name

```
        {
                -----
        public:
                ------
        friend datatype functionname(args);
};
```

Dynamic Memory Allocation

Static	Dynamic

Int data;
Float x;
Int array[2];
How much memory we need is know before the program is called static memory
	Many times we don't know how much memory we will need to store information.
	Size will be determined only at run time.
	New and delete operator are used to allocate memory

1. New operator: new creates a memory and returns memory address to the pointer.

```
syntax: new datatype;
eg: int *p; // int type pointer created
p= new int // p points to int type of memory
*p = 10;
cout<<*p;
```

We can aslo create an array.

```
int *p;
p = new int[5]; // p points to 5 locations of int type.
```

- Eg: include<iostream>
  using namespace std;
  Int main() {
- int *p;
- p= new int[5];
- p[0]=10;
- p[1]=20;
- p[2]=30;
- p[3]=40;
- p[4]=50;
- For(int i=0; i<5; i++;)
- {
- cout << p[i]<<endl;
- }
- Return 0;
- }

Delete operator:

- Dealocating a memory location of object.
- Syntax: delete pointer_variable;
    - Eg: delete p;
  - syntax for array: delete [size] pointer_variable;
    - Eg: delete[10] p;
- Eg:

```
#include <iostream>
Using namespace std;
Int main(){
        int *p;
        p=new int;
        *p=700;
        cout<<“value in p:”<<*p<<endl;
        delete p;
        cout<<“value after delete:”<<*p<<endl;
        return 0;
}
```

Eg: arrays

```
Include <iostream>
Using namespace std;
Int main(){
        int *p;
        p=new int [2]
        p[0]=10;
        p[1]=20;
        delete[]p;
        cout<<“p[0]”<<p[0]<<endl;
        cout<<“p[1]”<<p[1]<<endl;
        return 0;
}
```

This Pointer:

- Keyword this is used to represent an object that invokes a member function.
- This pointer points to the object for which this function was called.
- Eg: the function call A.max() will set the pointer this tp the address of object A.
- This pointer is automatically passed to a member function when it is called .
- Eg: class xyz

```
                {
                        int a;
                        --------
                };
```

We can also use it as, this -> a=20.

Static Data Members:

- A data member of a class can be static.
- Static member variable is similar to C static variable.
- Static member variable:
  - It is initialized to zero when the first object of its class is created
  - No other initialization is permitted.
  - Only one copy of that member is created for the entire class and is shared by all the objects of that class, no matter how many objects are created.
  - It visible only within the class, but its lifetime is the entire class.
- Static variables are normally used to maintain values common to the entire class.
- We can define static variables outside the class also.

Static member Functions:

- Like static member variable we can also have static member functions.
- Member function declared as static has the following properties
  - A static function can have access to only other static members declared in the same class.
  - A static member function can be called using the class name
  - Syntax: class name :: function name;

Constant Member Functions and Constant Objects,

- *Constant member function:*

  return_type function_name () **const** { *//function body* }

  ```
  public:
  void set_data(int a) { x = a; }
  ```

- class employee{

  public: // Members of the class go in here. };

  const employee Jake; // Defined employee object as an unmodifiable object

Friend class

- The syntax for implementing a friend class is:

```
        class ClassB;
        class ClassA {
                // ClassB is a friend class of ClassA
                friend class ClassB;
          ... .. ...
        }
class ClassB {
  ... .. ...
}
```

Practice problem

- https://www.scribd.com/document/517720302/MCQ-OOPS-WITH-C-QUESTIONS-WITH-ANSWER-converted