Object Oriented Programming using C++

Based on -

LECTURE NOTES: ON Object Oriented Programming Using C++

By:Dr. Subasish Mohapatra

Department of Computer Science and Application College of Engineering and Technology, Bhubaneswar

Created By: Jaydeep shah (24 March 2024) – For self-learning –radhey04ec@gmail.com

Three types of Programming language

1) **Machine code:** Directly execute by machines / CPU, No further conversion, Each Instructions perform very specific task, for examples load data / store data to memory area, or perform some mathematical operation by ALU.

Assembly means Human Readable Instructions like MOV, ADD etc

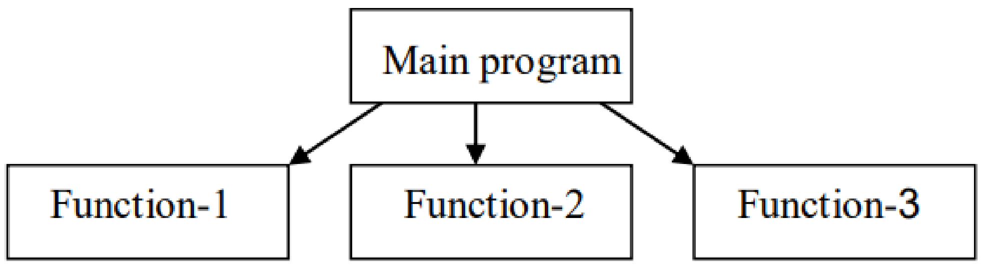
2) **Assembly Language (Low level):** This is very low level programming language, Very close to machine code and Hardware Architecture of machine / CPU, There is very strong correspondence (Ideally : one to one instruction) for directly context between Assembly instruction (Language) and Architecture's machine code. Assembler converts this code into executable machine code.
3) **High level programming language:** Very close to human language, easy to read and write, more focus on programming logic rather than hardware architecture. C / C++ /Java / Python all are High level programming language.

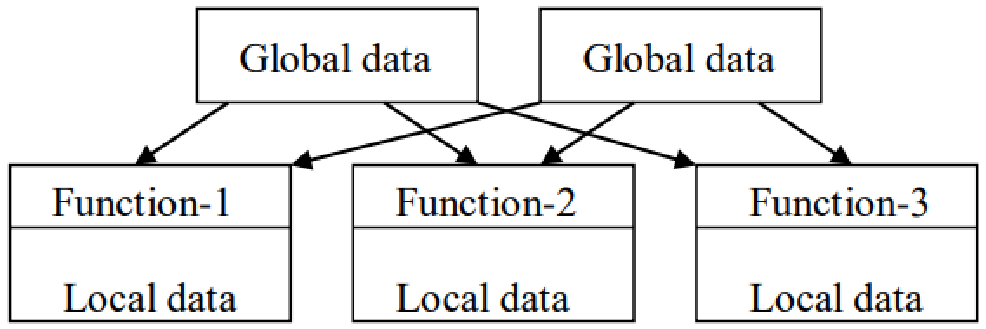
Further classification of High level programming language:

A) Procedure oriented programming (**POP)**

B) Object oriented programming (**OOP**)

A) **Procedure Oriented Programming**: Problem is viewed as sequence of thing. Creating set of instruction for performing tasks known as functions. More focus on process /function not on Data.





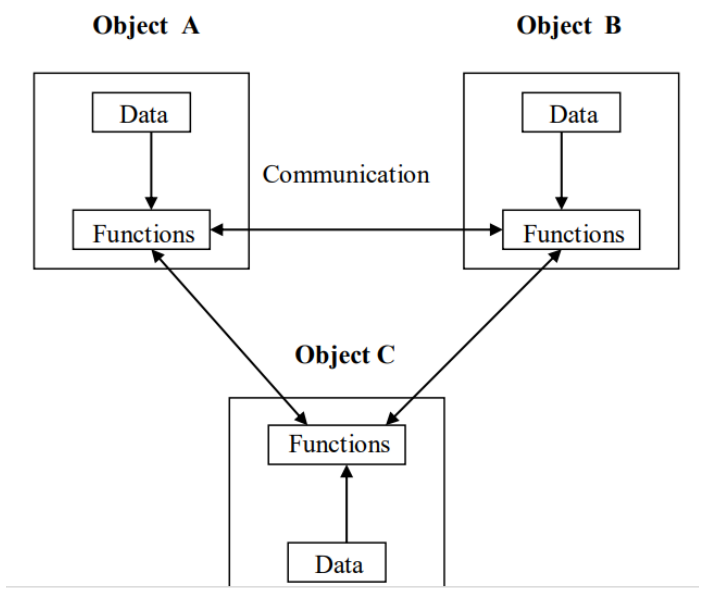
-Not more related to real world problem

-Top to Bottom Approach

-Data sharing between Functions / Programs

-No Data hiding.

B)**Object oriented programing**: More Focus on Data not procedure, Data and Function bind together and in future it use as template to create as on demand known as object.



-More closer to real world problem

-Bottom to top Approach

-Data not sharing between function / Data hiding possible.

-Problems divided into objects.

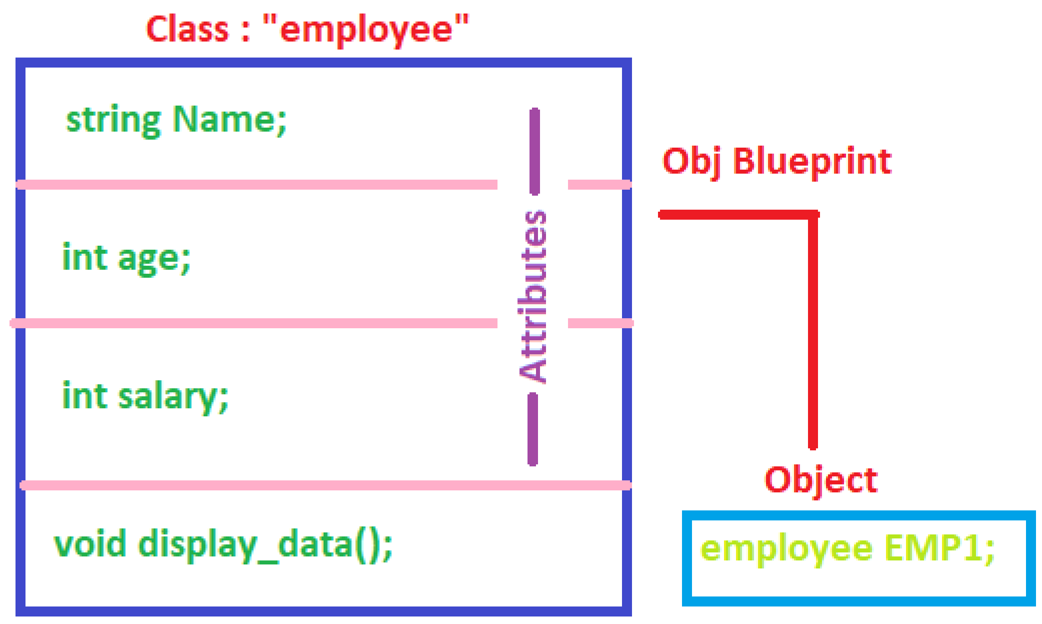
Concepts of OOPS:

1) **Object and Class:**

Main Idea behind OOP concept is combined Data and Programs or in other word combined data and functions. Class is blueprint or map for object. Class is group of Variables or Data-types and functions.

In C++ class is new data type / user defined data-type (Enhance version of structure from C). All objects from same class have same properties and attributes .This is way we can connect program to more real world problems.

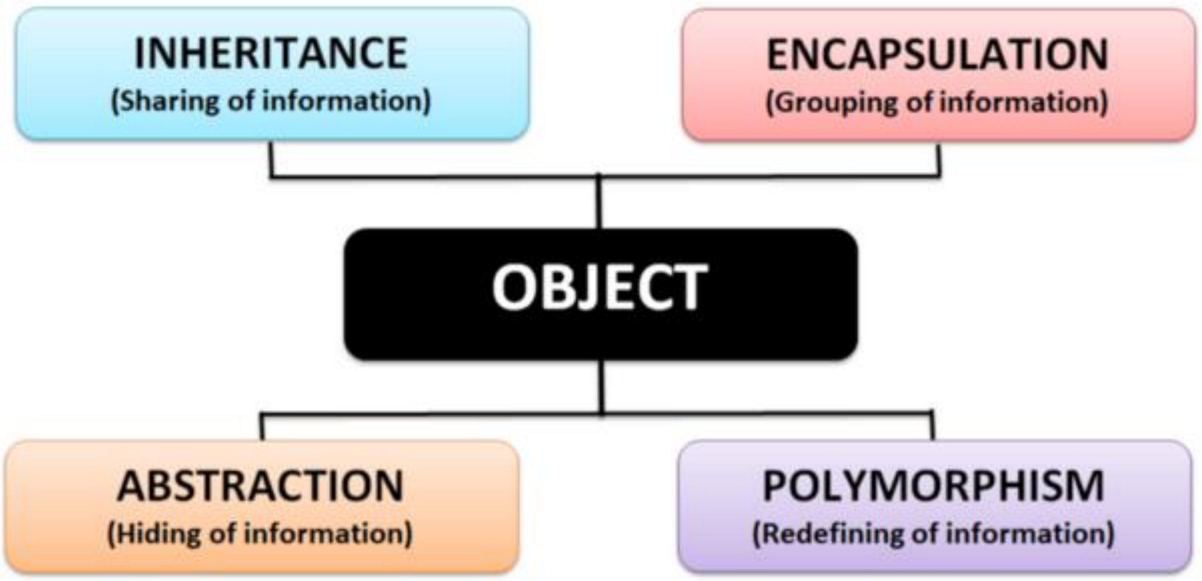
Example: We want to describe company's employee details. Employees have name, age , salary etc. All this thing / characteristics are use to describe employee details so we can use it in same group – class, it is known as **attributes of objects**. Same way to display details of the employee we may have to use function and these functions stored inside same class. Purpose of this function is only displaying data of employee so known as member function of the class.



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

//Create Class
class employess {
public:
       string Name;
       int    Age;
       int    Salary;

       void display_employee_details()
       {
              cout << "Name = " << Name << "Age = " << Age << endl;

       }
};
```

```
int main()
{
	//Create object
	employess emp1;
	emp1.Name = "Jaydeep";
	emp1.Age = 30;
	emp1.display_employee_details();
	return 0;
}
```

Inheritance = Reuse Information



Four pillars of OOP:

Access specifier (Public / Private / Protected))

1) **Abstraction**: Show only necessary/essential information to user, hiding background information. Class is example of Abstraction.
2) **Encapsulation**: Wrapping Data and Functions or operations into single unit known as class, Encapsulation means grouping the information. Objects which are not belonging to class can't access information or data for that class. Data only accessed by those function which are wrapped in that capsule or class. This functions provide interface between data and program logic.It provides security also.
3) **Inheritance:** It means reuse the information / sharing the information. Object of one class acquire properties of another class known as inheritance. We can add extra feature into old class without modifying it. (Parent and child class))
4) **Polymorphism**: Different form of same thing known as polymorphism. Polymorphism means "many forms", and it occurs when we have many classes that are related to each other by inheritance. **One function use with multiple use cases**. **Overloading is example of polymorphism**. It is able to express the operation of addition by a single operator say '+'. When this is possible you use the expression x + y to denote the sum of x and y, for many different types of x and y; integers , float and complex no. You can even define the + operation for two strings to mean the concatenation of the strings.

Other important concept is **Message passing**: Message passing is technique to transfer data / information between two or more objects of different classes. It is used when one object communicating with other object.

One object is request to use / invoke function (procedure) mentioned in other class's object. One object is sender, one object is receiver. Requesting object requires name of function, name of object and Argument values.

Example:

```
//Tutorial #2 - Message passing in C++
//Message passing - Communication between objects.
//Jaydeep Shah -Email: radhey04ec@gmail.com

#include <iostream>
using namespace std;


//Create one class
class Customer
{
public:
    void Update_Order_History()
    {
        cout << "Order history updated - done " << endl;
    }
};

//Create other class
class Order
{
public :
    void Send_Msg_to_Customer_Class(Customer* temp)
    {
        temp->Update_Order_History();
    }
};

int main()
{
    //Create object from Customer class
    Customer cust;

    //Create object of other class
    Order ord;

    //Requesting invoke of Customer method from Order class object
    ord.Send_Msg_to_Customer_Class(&cust);
}
```

Above is simple example of message passing.

SCOPE RESOLUTION(::) (NOTE – Definition outside class but declaration must be inside the class first)

It is possible to definition / function body may be outside of the class, using scope resolution operator. See below example.

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


//Create class
class people
{
	//Default Access is private
	char Name[30];
	int Age;

public:
	void details_entred(void);
	void display_details(void);
};

//Function defined ouside of class - using :: scope resolution
void people::details_entred()
{
	cout << "Enter Person name : ";
	cin >> Name;
	cout << endl;
	cout << "Enter Age : ";
	cin >> Age;
	cout << endl;


}

void people::display_details()
{
	cout << "Here is data - " << endl;
	cout << "Person name = " << Name << endl;
	cout << "Person Age is = " << Age << endl;

}

int main()
{
	//Create object
	people p1;
	p1.details_entred();
	p1.display_details();
	return 0;
}
```

OUTPUT:

```
Enter Person name : JAY

Enter Age : 30

Here is data -

Person name = JAY

Person Age is = 30
```

C and C++ both are block structured language. We know that same variable can be declared in different block ({ }) it act as local. If there are two variable with same name one is local and other is global then we can't access global variable from local block in C, But in C++ it is possible and here scope resolution :: operator is useful. See below example.

```
//Tutorial-6 - Scope resolution - Scope of varaible
//Jaydeep Shah - radhey04ec@gmail.com

#include <iostream>

using namespace std;

//Global Varaible m
int m = 10;

int main()
{
	//First Block
	int m = 20;

	//Second Block
	{
		int k = m;
		int m = 30;
		cout << "we are in inner block \n";
		cout << "k=" << k << endl;
		cout << "m=" << m << endl;
		cout << ":: m=" << ::m << endl;
	}
	cout << "\n we are in outer block \n";
	cout << "m=" << m << endl;
	cout << ":: m=" << ::m << endl;

	return 0;
}
```

OUTPUT:

```
we are in inner block

k=20

m=30

:: m=10

 we are in outer block

m=20

:: m=10
```

Terminology in C++

Token: Smallest element or unit in the program and meaningful for compiler known as token.

There are six types of token in C or C++

1) **Keywords:** Reserve words for programs like char, float, auto or void etc.

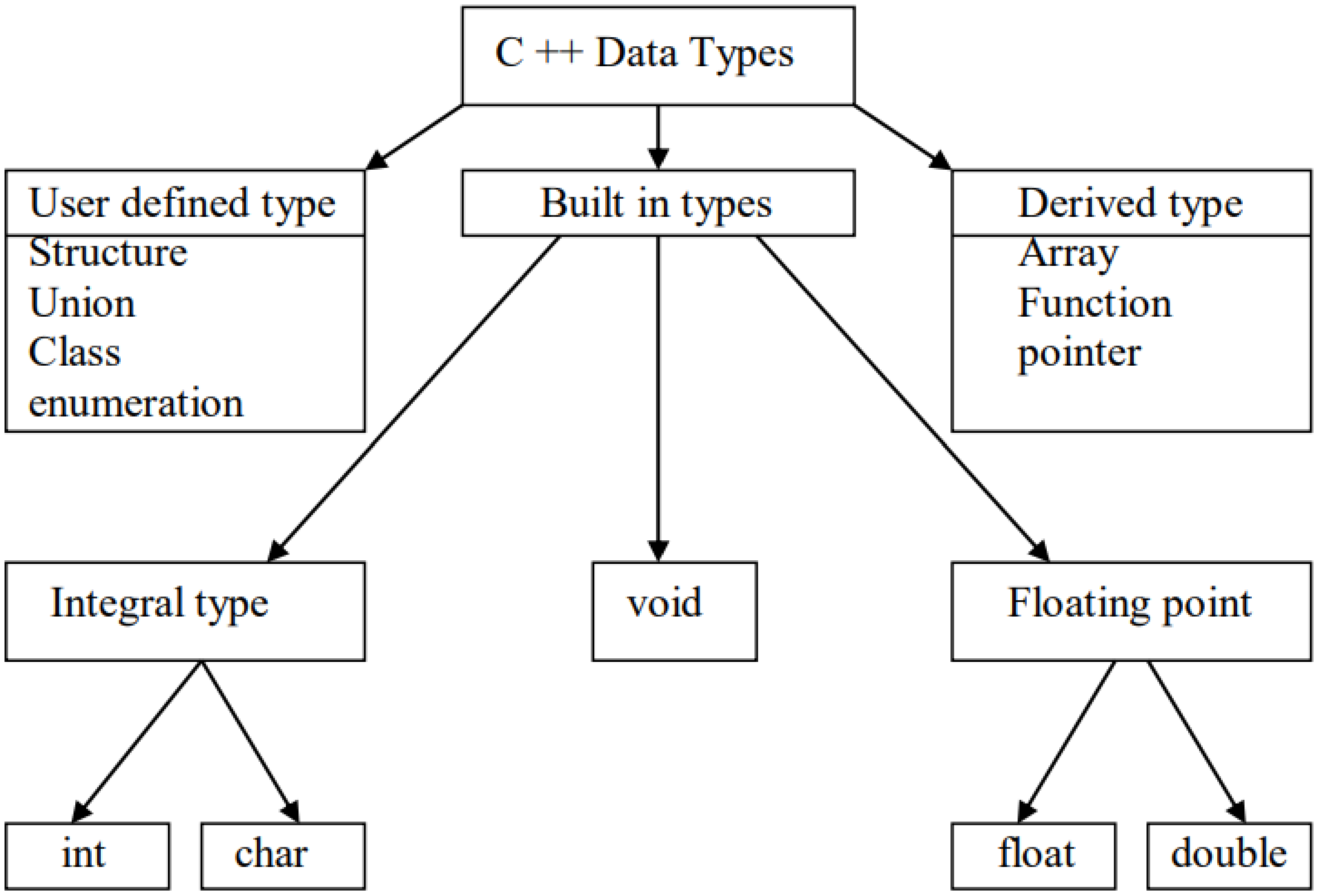
2) **Identifiers:** Name of variable, class, function, array etc. Every language has set of rules to use word as identifier.

3) **Constant:** Value remains unchanged throughout programs or not changes during execution.

4) **String:** Sequence of character inside "",In C++ string is string class use dynamic memory allocation.

5) **Operators:** it use to perform operation on data, it may be mathematical and logical operation in programs or for strings.

Datatypes:



We here only discussed enumerator.

Enumerator: User defined data-type, it is way to connect number with symbol or name for easy to use in code. We can assign any number with name but default start from zero (0).

```
//Tutorial-4 - Enumerator
//Jaydeep Shah - radhey04ec@gmail.com

#include <iostream>
using namespace std;

//Create enumerator
enum Gender
{
	MALE,
	FEMALE,
};


int main()
{

	//enume varaible
	Gender gender;
	gender = MALE;

	switch (gender)
	{
	case MALE:
		cout << "MALE";
		break;
	case FEMALE:
		cout << "FEMALE";
		break;
	}
	return 0;


}
```

It is also possible to assign random values in enumerator, or assign same value with multiple symbols
Example:

```
enum mix_data
{
a = 10,
b = 20,
c = 0,
d =1,
e =10,
};
```

Above enumerator is also valid.

<u>Note</u>: Any value assigned to enum variable which is not part of the definition of that variable will generate compile time error.

REFERENCE VARIABLE:

C++ supports OOP, and that's why **polymorphism** is pillar of this language.

C++ allow new interesting variable, you can alias old variable with new name known as reference variable. Alternative name.

But important thing is reference variable must be initialized during time of declaration.

Example:

int addition = 100;   //Original variable

int &sum = addition;   //Reference variable

Here is **"&" s**ign is overload with meaning of address operator.

Reference is something similar like Pointer, but not 100% same.

Here is why...
1) Reference must be initialize during declaration, For pointer it is not necessary.

2) We can reassigned pointer with different address pf different variable, in reference it is not possible.

Example:

```
//Tutorial-5 - Referece - Alais or use old varaible with new name (OOP - Polymorphism)
//Jaydeep Shah - radhey04ec@gmail.com

#include <iostream>
using namespace std;


//Original Variable
int addition = 100;
int& sum = addition;    //Reference varaible

int main()
{

       cout << "Value of addition = " << addition << endl;
       cout << "Value of sun =" << sum << endl;

       return 0;
}
```

Suppose there is function :
foo (int &n);

when function call foo(x);

At that time:
n is alias name of x, both have same memory address.
>> So x is not copy in n (no stack overload)
>> n address = x address

Extra Notes:

>> In C, main function may be void, but in C++ it returns integer value after execution to OS.

Inline function: it is like #define, pre-processor replace inline function where it call from to reduce call cost/ stack operation and speed.

Function is expand when it is invoke/call. This inline function is used when function is very small in size.

“inline” prefix with function name is required during definition.

Note: It is not necessary always inline function replace/expand during compile time, in some case it is act like normal function.

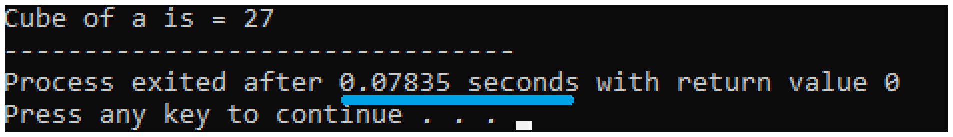
When inline function is large in size of memory, or when it is recursive, or when doesn’t return anything but return statement is there, or function have loop statement like for, while etc or switch or goto statement. It act like normal function.

Example :

```
//Tutorial-7 Inline Function - Pre processor/Compiler replace function call with function body
//Ideally use for small function
//Jaydeep Shah - radhey04ec@gmail.com

#include <iostream>

using namespace std;

//Inline function
inline double cube(double a)
{
        return (a * a * a);
}

int main()
{

        int a = 3;
        cout << "Cube of a is = " << cube(a);
        return 0;
}
```

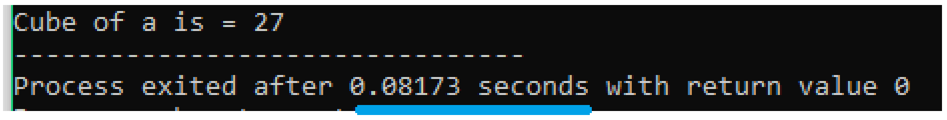
Execution time / cost of speed

When function is with inline prefix



```
Cube of a is = 27
--------------------------------
Process exited after 0.07835 seconds with return value 0
Press any key to continue . . .
```

Normal same function /without inline



```
Cube of a is = 27
--------------------------------
Process exited after 0.08173 seconds with return value 0
```

Default Argument in C++ (OOP):

During function call, users need to pass one or many argument for further procedure. But in many cases we need to use some default value of parameter for further processing. C++ allows using default argument value during function declaration.

Because of this default argument feature, we do not need to write two different functions, we can combine with single function with default argument value.

Example: We have one function for calculating interest.

Float compute_interest(int period, double amount,double interest_rate = 7.5)

User can enter interest rate ,but in case of absence of this detail function will consider this value is 7.5,because of default value.

```
//Tutorial-8 Default Argument functionality in C++
//Jaydeep Shah - radhey04ec@gmail.com

#include <iostream>
using namespace std;

//Function declaration
double compute_intr(float period, double amount, double intr_rate = 7.3);

int main()
{

	//Local variables for storing user details
	double INR = 0, rate = 0;
	float  YEAR = 0;

	cout << "Enter Your Amount " << endl;
	cin >> INR;
	cout << endl << "Enter Period in Year" << endl;
	cin >> YEAR;
	cout << endl << "Computing INTR = " << compute_intr(YEAR, INR);  //Function call with default Argument value
	cout << endl << "We consider INTR rate = 7.3%" << endl;
	cout << "Enter Your rate here : ";
	cin >> rate;
	cout << endl << "New INTR = " << compute_intr(YEAR, INR, rate); //Function call without default Argument
	return 0;
}


//Function definition (function body)
//Here no default Argument required,only at declaartion time
double compute_intr(float period, double amount, double intr_rate)
{
	return((period * amount * intr_rate) / 100);
}
```

NOTE: Position of default argument is at right side to left side, We cannot use default argument at middle position of function definition or declaration.

CONSTANT ARGUMENT:

Sometimes we will have to pass pointer or variable address as an argument to function. That time there is possibility of changes original value by function during computation or further process. Constant Argument is useful in this case, it does not allow changing argument parameter value, we can only use it for procedure using help of local variables, but we cannot modify it.

```
//Tutorial-9 Constatnt Argument in function
//Useful when we want to only process / compute something without modification original data
//Useful when Address or pointer will pass to function
//Jaydeep Shah - radhey04ec@gmail.com

#include <iostream>
#include <string>               //String operation library

using namespace std;


//Constant Argument inside function
//We can not change either length or string pointer, we can only proceed with using local varaible inside function
void print_my_data(const char* point, const int length)
{
      int count = 0;
      while (count < length)
      {
            cout << *(point + count);
            count++;
      }

      //Uncomment below - it will genrate compile time error
      //length++;        //Error = Modification of const argument varaible

}

int main()
{
      //Store predefined string in chaarcter Array
      char str_data[] = "This is constant argument example";

      //Function call
      print_my_data(str_data, strlen(str_data));

      return 0;
}
```

Note: In case of function try to change const argument, compiler will generate the error – modification of const variable not allow.

Function Overloading:

C++ supports OOP's polymorphism; you can use same function with multiple purposes. We can use same function name to perform variety of task. This is also known as **function overloading**.

We consider it as group of the functions (Family of function) with same function name but different type of argument.

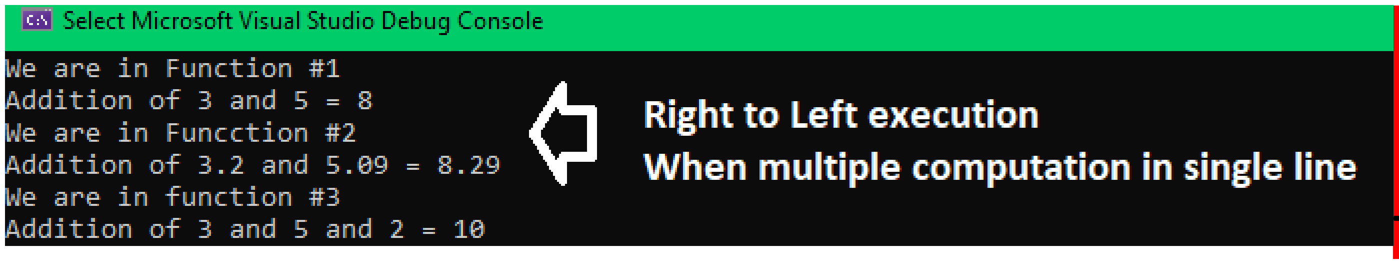
>> Compiler first tries to find exact match of proto function with call.

>> If exact match not found then compiler use integral promotion, convert char to int and float to double.

>> If after conversion no exact match found, compiler will generate error message.

```
//Tutorial-10 Function overloading (Polymorphisam - OOP)
//Same function name with different arguments known as functional overloading
//Jaydeep Shah - radhey04ec@gmail.com

#include <iostream>
using namespace std;

//All function family with same name "ADD"
//Function 1 with integer type argument
int ADD(int x, int y)
{
	cout << "We are in Function #1 " << endl;
	return (x + y);
}

//Function 2 with double type Argument
double ADD(double x, double y)
{
	cout << "We are in Funcction #2" << endl;
	return(x + y);
}

//Function 3 with different number of arguments
int ADD(int x, int y, int z)
{
	cout << "We are in function #3" << endl;
	return(x + y + z);
}

int main()
{
	//Execution flow of multiple instructions are Right to Left
	cout << "Addition of 3 and 5 = " << ADD(3, 5) << endl;
	cout << "Addition of 3.2 and 5.09 = " << ADD(3.2, 5.09) << endl;
	cout << "Addition of 3 and 5 and 2 = " << ADD(3, 5, 2) << endl;
	return 0;
}
```

OUTPUT:



```
We are in Function #1
Addition of 3 and 5 = 8
We are in Funcction #2
Addition of 3.2 and 5.09 = 8.29
We are in function #3
Addition of 3 and 5 and 2 = 10
```

Right to Left execution
When multiple computation in single line

This is only valid for printing statement in call function, return value follow order as per call position.

Extra Notes:

>>By default all members of class are private.

>>Private variables and private function only accessed by class member function.

>>Member function may be declare and define in class but it is also possible to define member function to outside of the class using "::" scope resolution operator.

Example:

```
class abc

{

public:

void test(void); //Function only declare inside class

};

//Function define outside the class

void abc :: test(void)

{

cout << "This is test";

}
```

>>Private member function only accessed by public member function of same class, private member function not possible to access by object using dot operator.

Array with class and Object:

Note: Do not use white space with “cin >>” . We will use getline() in future for solving this.

```
//Tutorial-11 Array with class and object
//Jaydeep Shah - radhey04ec@gmail.com

#include <iostream>
using namespace std;

//Create global class to store employee data
class empl_cls
{
	//Default data is private
	char Name[30];			//Array to store string type character data
	int  Age;
	int  Emp_Code;
	int  Salary;

	//Public method of access employee data
public:
	void enter_emp_data(void);
	void show_emp_data(void);

};

//Definition of class member methods
void empl_cls::enter_emp_data()
{
	cout << endl << "-------------------------------------" << endl;
	cout << "Name : ";
	cin >> Name;
	cout << endl << "Employee code = ";
	cin >> Emp_Code;
	cout << endl << "Age : ";
	cin >> Age;
	cout << endl << "Salary = ";
	cin >> Salary;
	cout << endl << "Detail Finish ...";
}

void empl_cls::show_emp_data()
{
	cout << endl << "****************************************" << endl;
	cout << "Emp code = " << Emp_Code << ", Name : " << Name << ", Age = " << Age << ", Salary : " << Salary << endl;
}

int main()
{
	//Suppose company have 3 employees
	empl_cls empl_details[3];			//Array as object of class

	//Eneter details one by one
	int k = 0;
	for (k = 0; k <= 2; k++)
	{
		empl_details[k].enter_emp_data();
	}
	cout << endl << "Employee details entered sucessfully....";

	//Details showing
	for (k = 0; k <= 2; k++)
	{
		empl_details[k].show_emp_data();
	}

	return 0;
}
```

Getline() instead of “cin >>” :

As we know “cin>>” not works with white space. For solving this we will have to use getline() function with white space enable. Please note that getline() only works with string data type not with character array[].

Check below example for more details:

```
//Tutorial-12 getline() function instead of cin >>, getline only work with string type
//cin object is not work with white space, alternative solution is getline()
//Jaydeep Shah - radhey04ec@gmail.com

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

class students
{

	string Name;
	int  RollNumber;
public:
	void enter_details()
	{
		cout << endl << "Student details : " << endl;
		cout << "Name = ";
		std::getline(std::cin >> std::ws, Name);   //getline only work with string type
		cout << endl << "RollNumber = ";
		cin >> RollNumber;
	}
	void show_details()
	{
		cout << endl << "Student Details : " << endl;
		cout << "Name = " << Name << "; RollNumber = " << RollNumber << endl;
	}
};

int main()
{

	//Create object
	students student[3];
	int k = 0;

	//Enter student details
	for (k = 0; k <= 2; k++)
	{
		student[k].enter_details();
	}

	cout << endl << "Details Filling completed" << endl;
	for (k = 0; k <= 2; k++)
	{
		student[k].show_details();
	}

	return 0;
}
```

Static data member / static variable inside class:

static variable or data member have certain characteristic, it act like global but accessible within scope of class. It’s lifetime is throughout the program. **Only one copy will be made of static variable, not depends on how many object we created**.

>> Also note that static variable is always initialize with zero automatically.

>> This static variable only possible to access by object of that class.

Example:

Always remember - static variable has only one copy

```
//Tutorial-13 static variable inside class
//Note that after declaration, definition of static variable also required.
//Jaydeep Shah - radhey04ec@gmail.com

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

class abc
{
public:
	static int stc_var;		//Static varaible declaraion inside class
	int x;

};


//static varaible defined - This must required
int abc::stc_var;

int main()
{

	//Create First object
	abc obj1;
	obj1.stc_var = 10;
	obj1.x = 3;

	//Create second object
	abc obj2;
	cout << "Static varaible access by second object and value is : " << obj2.stc_var;


	return 0;
}
```

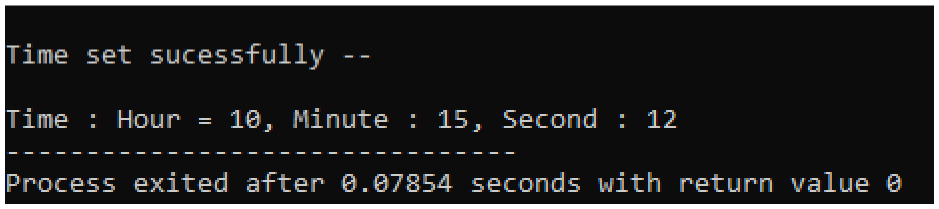
This is because: static variable life throughout code and only one copy of static variable will made. Define outside of static variable is necessary because we tell compiler, this class has static variable and memory will be define somewhere else.

PASS BY VALUE / PASS BY REFERENCE (OBJECT AS ARGUMENT):

1) **PASS BY VALUE:** One copy of entire object is send/pass as argument into function. Because we passed copy of object inside function any changes applied on object will not affect original object.
2) **PASS BY REFERENCE:** Object's address passed as argument inside function. Function will work on actual object.

Example:

```
//Tutorial-14 Pass by reference and pass by value

//Pass by ref : Object Address will pass to function, operation occure on original function
//Pass by val : Copy of original object pass as argument, original object remain unchange
//Jaydeep Shah - radhey04ec@gmail.com

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


//Create class
class set_get_time
{
	int Hour, Minute, Second;

public:

	//Pass by value example
	void set_clock_time(int h, int m, int s)
	{
		//Set varaible
		Hour = h;
		Minute = m;
		Second = s;
		cout << endl << "Time set sucessfully --" << endl;
	}
	//Pass by reference example
	void get_clock_time(set_get_time* t)
	{
		cout << endl << "Time : Hour = " << t->Hour << ", Minute : " << t->Minute << ", Second : " << t->Second;
	}
};


int main()
{

	//Create object
	set_get_time time;

	//Pass by value
	time.set_clock_time(10, 15, 12);  //Set time 10:15:12

	//Pass By reference - Get Time
	time.get_clock_time(&time);  //Obj Address pass

	return 0;
}
```



```
Time set sucessfully --

Time : Hour = 10, Minute : 15, Second : 12
--------------------------------
Process exited after 0.07854 seconds with return value 0
```

Friend function:

Friend function is non member function, but possible to access private and protected variable.

Suppose we have two class 1) Scientist 2) Engineers. Both class have own attributes and variables and own member function. As we knew that private data of class are only accessed by their member function only.

But sometimes some data need to access by outer function from class, suppose both have income_tax variable, and we want to access this value by using function which is not part of any of the class. That time C++'s "friend function" concept is very useful.

Using friend function we can access private data of that calss, and we can declare same friend function to any number of classes.

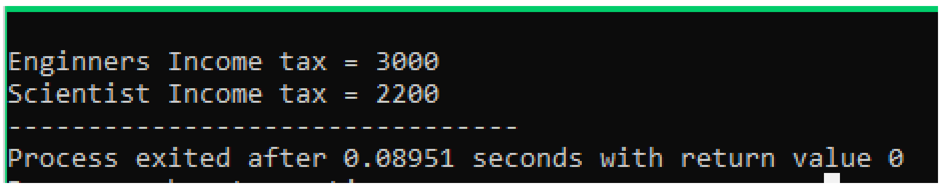
For friend function, need to declare in both class with friend prefix.

Definition of friend function may be anywhere without friend prefix or scope :: resolution.

```
//Tutorial-15 Friend function.
//Friend function is not part of class, but possible to access private variable of any class (Function must be declare as friend inside
class).
//Jaydeep Shah - radhey04ec@gmail.com

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


//Class creation
class Scientist;                //Declaration - For avoiding error in friend function - it is using this class argument

class Engineers
{
public:
	int income_tax_amount;

	friend void print_income_tax_amount(Engineers E, Scientist S, int n); //Friend function declaration
};

//Class creation (Definition)
class Scientist
{
public:
	int income_tax_amount;

	friend void print_income_tax_amount(Engineers E, Scientist S, int n); //Friend function declaration
};


//Friend function body - Definition
void print_income_tax_amount(Engineers E, Scientist S, int n)
{
	switch (n)
	{
	case 1:
		cout << endl << "Enginners Income tax = " << E.income_tax_amount;
		break;

	case 2:
		cout << endl << "Scientist Income tax = " << S.income_tax_amount;
		break;

	default:
		cout << "Wrong Argument" << endl;
	}
}
```

Friend function is non member function, so for accessing private variable you will have to pass that object (class as argument always required)

```
int main()
{

	//Create object
	Engineers Eng;
	Scientist Sci;

	//Set income tax value
	Eng.income_tax_amount = 3000;
	Sci.income_tax_amount = 2200;

	print_income_tax_amount(Eng, Sci, 1);
	print_income_tax_amount(Eng, Sci, 2);

	return 0;
}
```



```

Enginners Income tax = 3000
Scientist Income tax = 2200
--------------------------------
Process exited after 0.08951 seconds with return value 0
```