C++ Programming

Trainer : Pradnyaa S. Dindorkar

Email: pradnya@sunbeaminfo.com

Todays Topics

1. namespace
2. cin and cout
3. complex class
4. Modular Approach
5. Constant
6. References
7. Difference between Pointers and reference

1. Sum function and Copy Constructor
2. New and delete
3. Difference between New and malloc
4. Deep copy and shallow copy
5. static variable and function
6. Friend function

Sum function and Copy Constructor

- Copy constructor is a single parameter constructor hence it is considered as parameterized constructor
- Example: sum of two complex number

  Complex sum(const Complex &c2)

Copy constructor

Complex c2(c1)

or

Complex c2=c1

$$\begin{tikzcd} C1 \arrow[r] & 7 \arrow[d] & + j & 6 \arrow[d] \\ C2 \arrow[r] & 7 & + j & 6 \end{tikzcd}$$

Write a function in complex class to add 2 complex numbers

$$\begin{array}{l} C1 \rightarrow 7+j6 \\ \quad + \\ C2 \rightarrow 3+j2 \\ \hline C3 \rightarrow 10+j8 \end{array}$$

Dynamic Memory Allocation

- If we want to allocate memory dynamically then we should use new operator and to deallocate that memory we should use delete operator.
- If pointer contains, address of deallocated memory then such pointer is called dangling pointer.
- When we allocate space in memory, and if we loose pointer to reach to that memory then such wastage of memory is called memory leakage.

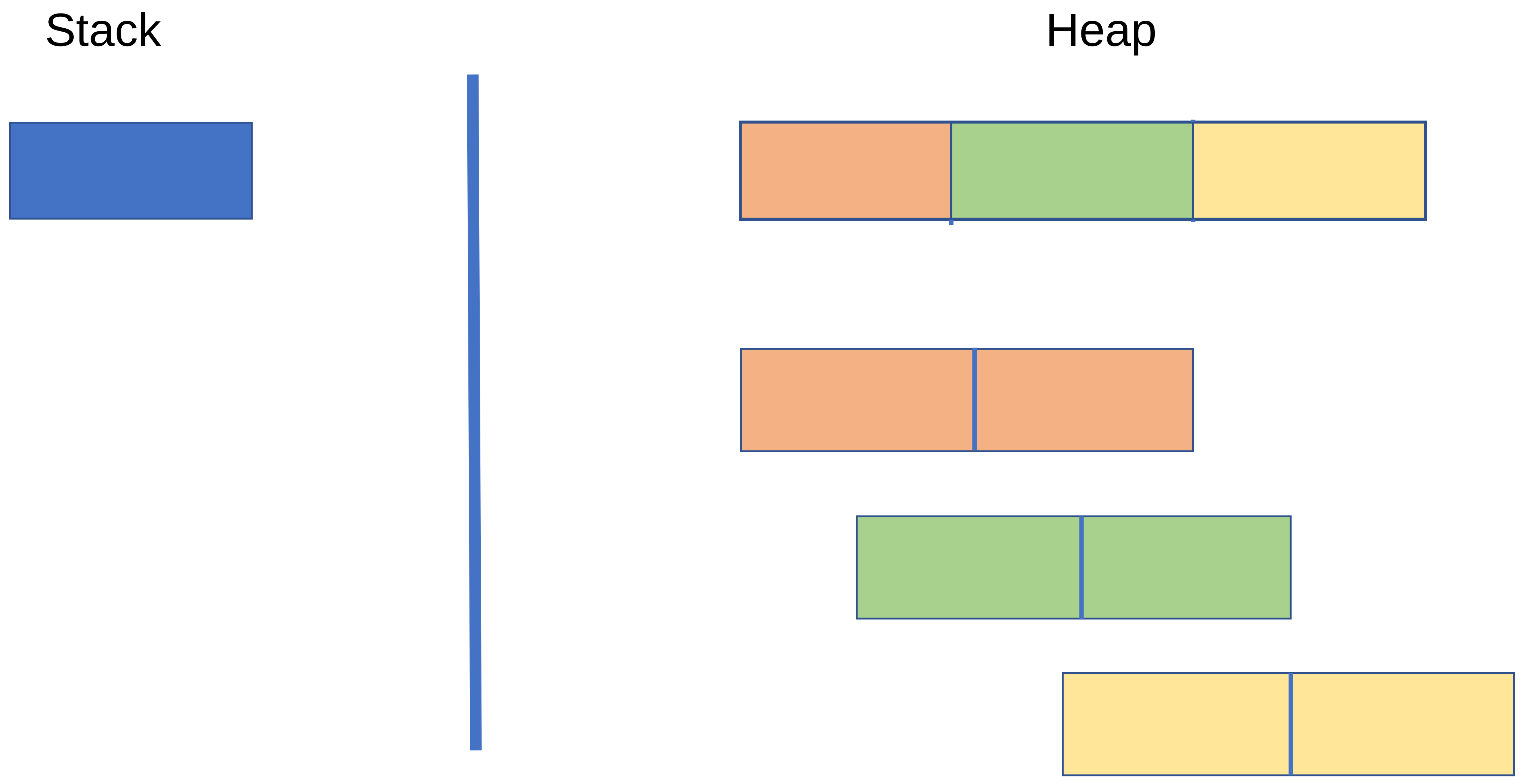
- Example :

```
int main()
{
    int *ptr = new int;           //int *ptr = ( int* )::operator new( sizeof( int ) * 1 );
    *ptr = 125;                   //Dereferencing
    cout<<"Value   :              "<<*ptr<<endl; //Dereferencing
    delete ptr;                   //::operator delete( ptr );
    ptr = NULL;
    return 0;
}
```

Allocate memory for 2D array using new and delete

	Column [0]	Column[1]
Row [0]	**Arr[0][0]**	**Arr[0][1]**
Row [1]	**Arr[1][0]**	**Arr[1][1]**
Row [2]	**Arr[2][0]**	**Arr[2][1]**

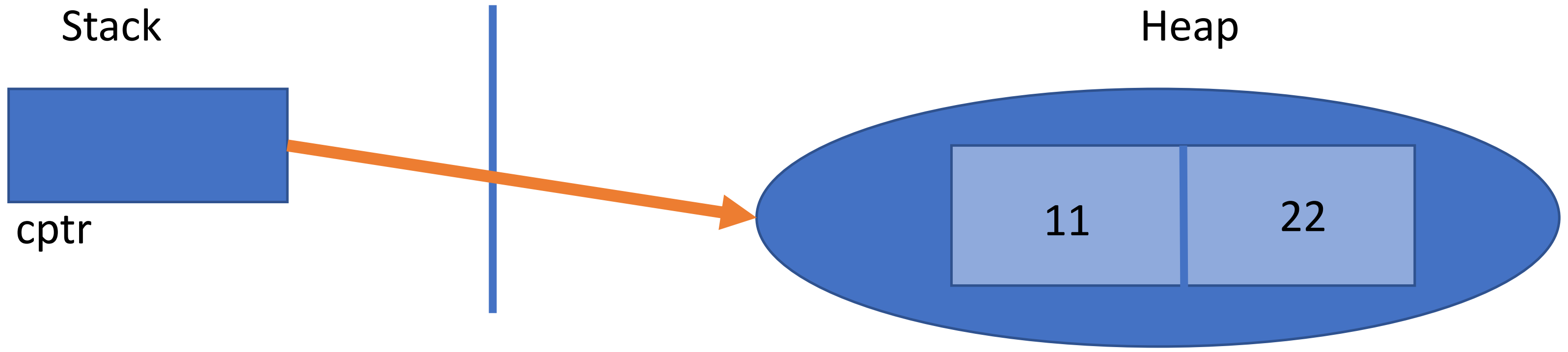
Allocate memory for 2D array using new and delete



Heap Based object

```
Complex *cptr=new Complex (11,22) ;
delete cptr;
```

- By using new we are allocating dynamic memory for complex class object .
- Object get created on heap section hence this object is call Heap based or dynamic object.
- Cptr is complex type pointer which is holding the address of that dynamic object.



Difference between malloc and new

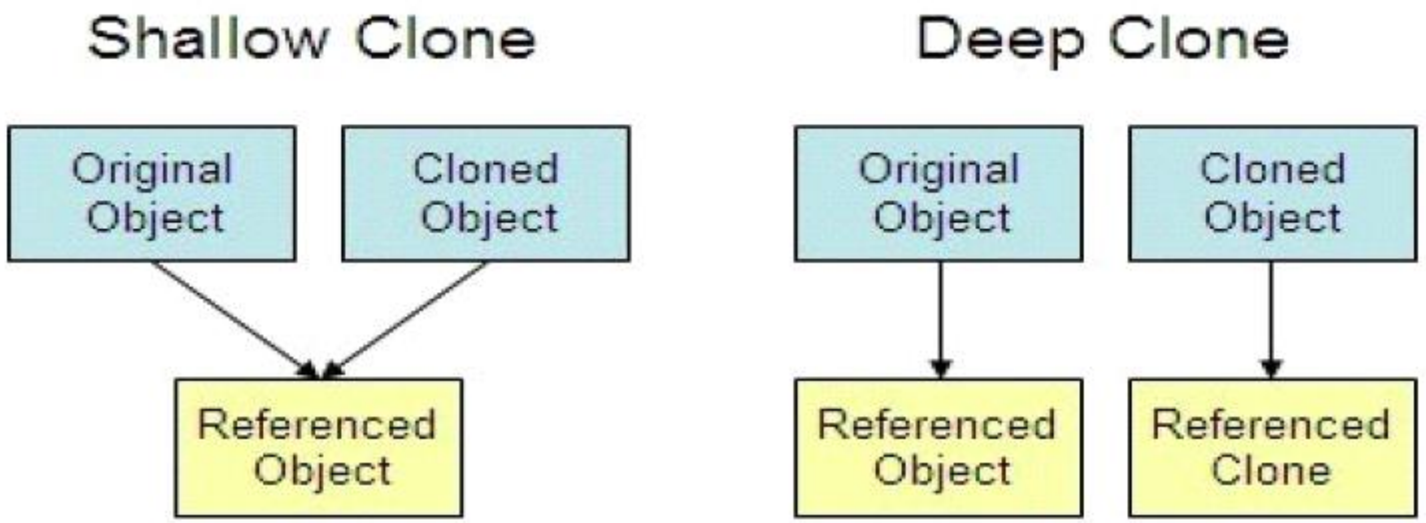
new	malloc
new is an operator.	malloc is a function.
new returns a typecasted operator, so no need to do explicit typecasting.	malloc returns void pointer,malloc returns void pointer,cast it explicitly, before use.
We must mention the datatype while allocating the memory with new.	malloc accepts only the exact no. of bytes required, so no need to mention datatype.
When memory is allocated with new, constructor gets called for the object.	When memory gets allocated by malloc function, constructor function does not gets called.

Difference between malloc and new

new	malloc
Memory allocated by new is released by the operator delete.	Memory allocated by malloc is released by function free.
Destructor is called when memory is released with delete.	Destructor is NOT called when the memory is released with free.
To release memory for array syntax is delete[ ]	To release memory for array syntax is free(ptr);
In case new fails to allocate memory, it raises a Run time exception called as bad_alloc.	In case malloc fails to allocate memory it returns a NULL.

Object Copying

- In object-oriented programming, “object copying” is a process of creating a copy of an existing object.
- The resulting object is called an object copy or simply copy of the original object.
- Methods of copying:
  - Shallow copy
  - Deep copy



Types of Copy

- **Shallow Copy**
  - The process of copying state of object into another object.
  - It is also called as bit-wise copy.
  - When we assign one object to another object at that time copying all the contents from source object to destination object as it is. Such type of copy is called as shallow copy.
  - Compiler by default create a shallow copy. Default copy constructor always create shallow copy.
- **Deep Copy**
  - Deep copy is the process of copying state of the object by modifying some state.
  - It is also called as member-wise copy.
  - When class contains at least one data member of pointer type, when class contains user defined destructor and when we assign one object to another object at that time instead of copy base address allocate a new memory for each and every object and then copy contain from memory of source object into memory of destination object. Such type of copy is called as deep copy.

Shallow Copy

- Process of copying state of object into another object as it is, is called shallow copy.
- It is also called as bit-wise copy / bit by bit copy.
- Following are the cases when shallow copy taken place:
    1. If we pass variable / object as a argument to the function by value.
    2. If we return object from function by value.
    3. If we initialize object: Complex c2=c1
    4. If we assign the object , c2=c1;
    5. If we catch object by value.
- <u>Examples of shallow copy</u>

  Example 1: (Initialization)

  ```
  int num1=50;
  int num2=num1;
  ```

  Example 2: (Assignment)

  ```
  Complex c1(40,50);
  c2=c1;
  ```

Deep Copy

- It is also called as member-wise copy. By modifying some state, if we create copy of the object then it is called deep copy.
    - Conditions to create deep copy
        - Class must contain at least one pointer type data member.

```
class Array
		{
			private:
				int size;
				int *arr;
				//Case - I
				public:
				Array( int size )
				{
					this->size = size;
					this->arr = new int[ this->size ];
				}
		};
```

- Steps to create deep copy
    - 1. Copy the required size from source object into destination object.
    - 2. Allocate new resource for the destination object.
    - 3. Copy the contents from resource of source object into destination object.

Static Variable

- All the static and global variables get space only once during program loading / before starting execution of main function
- Static variable is also called as shared variable.
- Initialized static and global variable get space on Data segment.
- Default value of static and global variable is zero.
- Static variables are same as global variables but it is having limited scope.

Static Methods or Static Member Functions

- Except main function, we can declare global function as well as member function static.
- To access non static members of the class, we should declare member function non static and to access
- static members of the class we should declare member function static.
- Member function of a class which is designed to call on object is called instance method. In short non static member function is also called as instance method.
- To access instance method either we should use object, pointer or reference to object.
- static member function is also called as class level method.
- To access class level method we should use classname and ::(scope resolution) operator.
- This pointer is not available in static member function .

Friend :-

- A non member function of a class which designed to access private data of a class is called friend function.

- To declare a function as a friend of a class, precede the function prototype in the class definition with keyword **friend**

```
class MyData
{
private:
            int pin;
            int pass;

public:
            MyData(int pin,int pass);
            void PrintMyAccDetails();
            friend void anyFunction();
};
```

```
void anyFunction()
{
            MyData d1;
            d1.pass=9898;
            d1.pin=9999;
            d1.PrintMyAccDetails();
}
```

C++ is not a pure Object Oriented Language Because

- You can write code without creating a class in C++, and main() is a global function.
- Support primitive data types, e.g., int, char, etc. Instances(variable) of these primitive types are NOT objects.
- C++ provides "Friend" which is absolute corruption to

  the OO-Principle of encapsulation.

Operator Overloading

- operator is token in C/C++.
- It is used to generate expression.
- operator is keyword in C++.
- Types of operator:
    - Unary operator (++,--,&,!,~,sizeof())
    - Binary Operator (Arithmetic, relational, logical , bitwise, assignment)
    - Ternary operator (conditional)
- In C++, also we can not use operator with objects of user defined type directly.
- If we want to use operator with objects of user defined type then we should overload operator.
- To overload operator, we should define **operator function.**
- **We can define operator function using 2 ways:**
    - **Using member function**
    - **Using non member function**

Need Of Operator Overloading

- we extend the meaning of the operator.
- If we want to use operator with the object of use defined type, then we need to overload operator.
- To overload operator, we need to define operator function.
- In C++, operator is a keyword
  - Suppose we want to use plus(+) operator with objects then we need to define operator+( ) function.

We define operator function either inside class (as a member function) or outside class (as a non-member function).	Point pt1(10,20), pt2(30,40), pt3;
	
	pt3 = pt1 + pt2; //pt3 = pt1.operator+(pt2); //using member function
	//or
	pt3 = pt1 + pt2; //pt3 = operator+(pt1, pt2); //using non member function

Operator Overloading

using member function

- **operator function must be member function**
- If we want to overload, binary operator using member function then **operator function should take only one parameter.**
  - Example : c3 = c1 + c2; //will be called as ------ c3 = c1.operator+(c2)

  Example :

```
Point operator+( Point &other ) //Member Function
	{
	Point temp;
	temp.xPos = this->xPos + other.xPos;
	temp.yPos = this->yPos + other.yPos;
	return temp;
	}
```

using non member function

- **Operator function must be global function**
- If we want to overload binary operator using non member function then **operator function should take two parameters.**
  - **Example :** c3 = c1 + c2; //will be called as ------c3 = operator+(c1,c2);

Example:

```
Point operator+( Point &pt1, Point &pt2 ) //Non Member Function
{
Point temp;
temp.xPos = pt1.xPos + pt2.xPos;
temp.yPos = pt1.yPos + pt2.yPos;
return temp;
}
```

We can not overloading following operator using member as well as non member function:

1. dot/member selection operator(.)
2. Pointer to member selection operator(.*)
3. Scope resolution operator(::)
4. Ternary/conditional operator(? :)
5. sizeof() operator
6. typeid() operator
7. static_cast operator
8. dynamic_cast operator
9. const_cast operator
10. reinterpret_cast operator

We can not overload following operators using non member function:

- Assignment operator(=)
- Subscript / Index operator([])
- Function Call operator[()]
- Arrow / Dereferencing operator(→)

MCQ

Q : Pick the correct statement about references in C++.

a) References stores the address of variables

b) References and variables both have the same address

c) References use dereferencing operator(*) to access the value of variable its referencing

d) References were also available in C

MCQ

Q : Pick the correct statement about references in C++.

a) References stores the address of variables

b) References and variables both have the same address

c) References use dereferencing operator(*) to access the value of variable its referencing

d) References were also available in C

Q. By default how the value are passed in c++?

a) call by value
b) call by reference
c) call by pointer
d) call by object

Q. By default how the value are passed in c++?

a) call by value

b) call by reference

c) call by pointer

d) call by object

Q. How are the constants declared?

a) const keyword

b) #define preprocessor

c) both const keyword and #define preprocessor

d) $define

Q. How are the constants declared?

a) const keyword

b) #define preprocessor

c) both const keyword and #define preprocessor

d) $define

Q. Can a constructor function be constant?

a) Yes, always

b) Yes, only if permissions are given

c) No, because objects are not involved

d) No, never

Q. Can a constructor function be constant?

a) Yes, always

b) Yes, only if permissions are given

c) No, because objects are not involved

d) No, never

Thank You