19CSE201 :Advanced Programming

Lecture 13
Memory Management in C++

By
Ritwik M
Assistant Professor(Sr)
Dept. Of Computer Science & Engg
Amrita Vishwa Vidyapeetham - Coimbatore

Based on material from Stroustrup B. Programming: principles and practice using C++. Second edition, Addison Wesley; 2014 and multiple sources on the internet

A Quick Recap

- OOP Overview
- Abstraction
- Encapsulation
- Inheritance
- Examples

Memory Management - Types

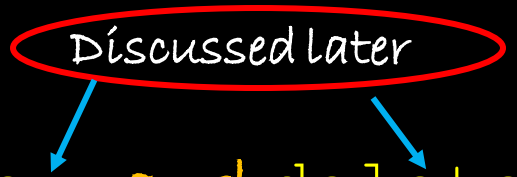
- Static Memory Allocation
- Dynamic Memory Allocation

Memory Management - Static Allocation

- Memory allocated for variables during compile time itself.
- Once allocated it cannot be either expanded or reduced at a later point in time.
- The memory size is known before compile time and cannot be altered during compile time.
- Example `int a[10]`

Memory Management - Dynamic Allocation

- Helps in optimizing memory usage and easier to modify memory allocation



- Carried out in C++ using the `new` and `delete` operators

- Allocates and frees memory dynamically during runtime

But First... Pointers!!

Pointers in C++

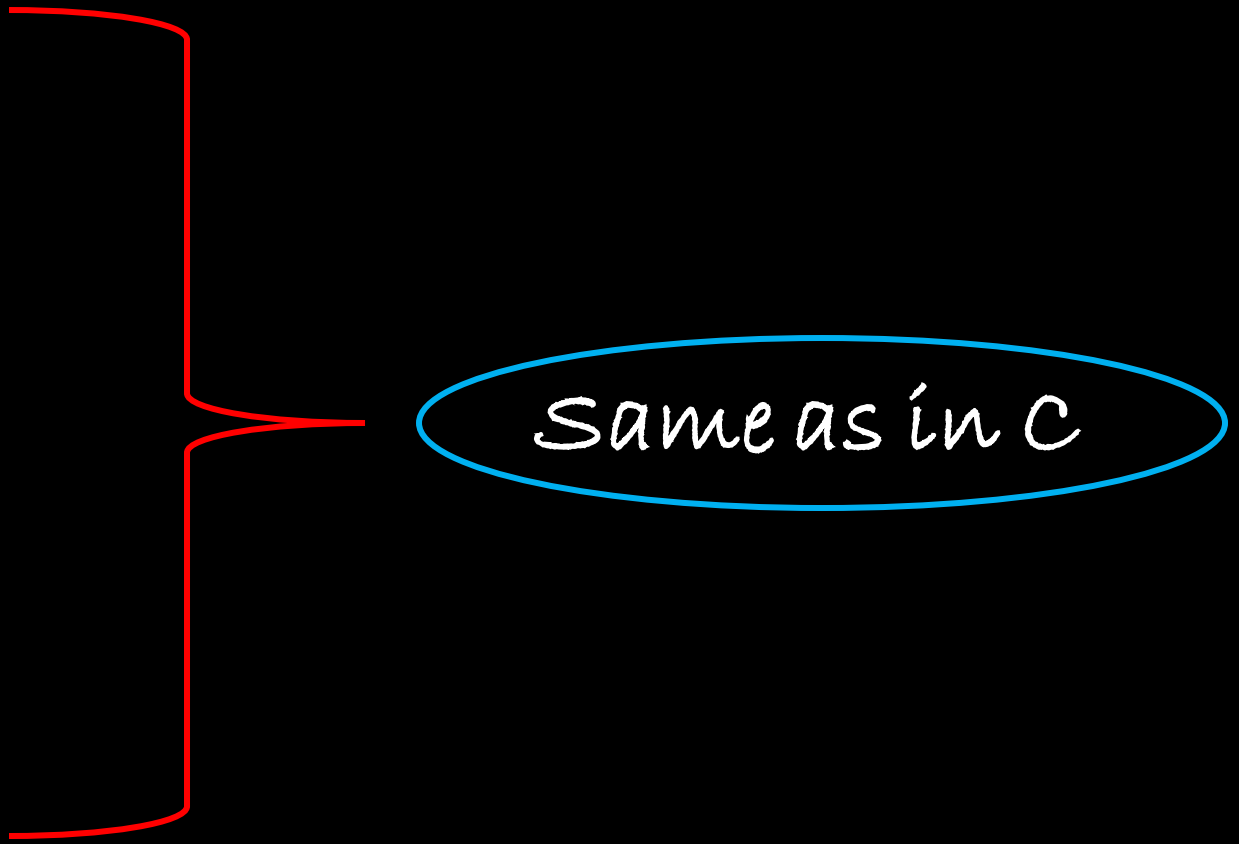
- Dynamic allocation not possible without knowledge of memory location for accessing data!
- Usage as done in C
- Declaration:
  - `Type *variableName`
  - Where type is one of the C++ datatypes.
- Process:
  - Define the pointer variable
  - Assign the address of a variable to the pointer variable
  - Access the value at the address in the pointer variable

Example

```
int main () {
// actual variable declaration.
   int  var = 20;
// pointer variable
   int  *ip;

// store address of var in pointer
variable
   ip = &var;
   cout << "Value of var variable: ";
   cout << var << endl;
   // print the address stored in
   //ip pointer variable

   cout << "Address stored in ip
variable: ";
   cout << ip << endl;

   // access the value at the address
   // available in pointer

   cout << "Value of *ip variable: ";
   cout << *ip << endl;


   return 0;
  }
```

Other concepts in pointers - Recap

- Null pointer
- Pointer Arithmetic
- Pointers and Arrays
- Array of pointers
- Pointer to pointer
- Passing pointers to functions
- Return pointer from functions



Additional concepts in C++ pointers

- Pointer to class
- Pointers to data members
- Pointers with Objects
- "this" pointer
- Pointers to member functions
- Pointer Applications using New and Delete

Pointer to a Class

- Is done exactly the same way as a pointer to a structure
- to access members of a pointer to a class, make use of the member access operator (->)
- Also, keep in mind that the pointers must first be initialized before using
- Scenarios
  - Pointers to data member of a class
  - Pointers to member functions

Example – Pointer to a Class

```
class Simple{
    public:
    int a;
};

int main(){
    Simple obj;
    Simple* ptr;   // Pointer of class type
    ptr = &obj;

    cout << obj.a;
    cout << ptr->a;  // Accessing member with pointer
}
```

Pointer to Data Member of class

- Syntax for Declaration :
  - `datatype class_name :: *pointer_name;`

- Syntax for Assignment:
  - `pointer_name = &class_name :: datamember_name;`

- Both declaration and assignment can be done in a single statement also.
  - `datatype class_name::*pointer_name = &class_name::datamember_name ;`

Using Pointer with Objects

- For accessing normal data members we use the dot (.) operator and arrow (->) for accessing class members
- When using a pointer to a data member, it has to be dereferenced to get what it is pointing to.

- Syntax for Declaration :
  - `objectName.*pointerToMember`
- Syntax for Assignment: - using the pointer to the object
  - `objectPointer -> *pointerToMember`
- The syntax is very tough; hence they are only used under special circumstances.

Example: Pointer to Data Member of a class

```
class Data
{
    public:
    int a;
    void print()
    {
        cout << "a is "<< a;
    }
};
```

```
int main()
{
    Data d, *dp;
    dp = &d;  // pointer to object
    // pointer to data member 'a'
    int Data::*ptr=&Data::a;

    d.*ptr=10;
    d.print();

    dp->*ptr=20;
    dp->print();
}
```

Pointer to Member Functions of Class

- Syntax for Declaration :
  - `returnType (className::*ptrName) (argumentType);`

- Both declaration and assignment can be done in a single statement also.
  - `returnType(className::*ptrName)(argumentType) = &className::functionName;`

Example: Pointer to Member Function of a class

```
class Data
{
    public:
    int f(float)
    {
        return 1;
    }
};
```

```
// Declaration and assignment
int (Data::*fp1) (float) = &Data::f;

// Only Declaration
int (Data::*fp2) (float);
int main(0
{
// Assignment inside main()
    fp2 = &Data::f;
}
```

Understanding "this" pointer

- Every object in C++ has access to its own address through an important pointer called "this" pointer.
- The "this" pointer is an implicit parameter to all member functions. Therefore, inside a member function, this may be used to refer to the invoking object.
- NOTE: Friend functions do not have a "this" pointer, because friends are not members of a class. Only member functions have a "this" pointer.

"this" pointer - Example

```
class Box {
   public:
      // Constructor definition
      Box(double l = 2.0, double b = 2.0, double h = 2.0) {
         cout <<"Constructor called." << endl;
         length = l;
         breadth = b;
         height = h;
      }
      double Volume() {
         return length * breadth * height;
      }
      int compare(Box box) {
         return this->Volume() > box.Volume();
      }

   private:
      double length;     // Length of a box
      double breadth;    // Breadth of a box
      double height;     // Height of a box
};
```

```
int main(void) {
   Box Box1(3.3, 1.2, 1.5);    // Declare box1
   Box Box2(8.5, 6.0, 2.0);    // Declare box2

   if(Box1.compare(Box2)) {
     cout << "Box2 is smaller than Box1" <<endl;
   }
   else {
      cout << "Box2 >= Box1" <<endl;
   }
   return 0;
}
```

Memory Management - Dynamic Allocation

- Also Called Dynamic Memory Allocation (DMA)
- Allocated on a Heap
- Uses:
  - Allocates memory of variable size.
  - Programmer freedom to allocate and de-allocate memory
  - Eg: Linked List, Tree, etc.
- Carried out in C++ using the `new` and `delete` operators

Ref. Data Structures course

The New Operator

- New is an operator in C++ used to allocate memory dynamically.
- Create an object in the heap during run time.
- It initializes the memory to the pointer variable and returns its address.
- Syntax: `pointerVariable = new datatype;`
- syntax to initialize the memory,
  `pointerVariable = new datatype(value);`
- syntax to allocate a block of memory,
  `pointerVariable = new datatype[size];`

Example – New Operator

```
int main ()
{
 int *ptr1 = NULL;
// allocate 4 bytes for an int type
 ptr1 = new int;

// allocate 4 bytes  float type and initialize to
the given value
 float *ptr2 = new float(223.324);

 // create an array of int of size 28.
 int *ptr3 = new int[28];
 *ptr1 = 25;
 cout << "Pointer variable 1 : " << *ptr1<<endl;
 cout << "Pointer variable 2 : " << *ptr2 << endl;
```

```
 if (!ptr3)
 {cout << "Memory Allocation failed\n"; }
 else
 {
  for (int i = 10; i < 15; i++)
    {  ptr3[i] = i+1;    }
  cout << "Value in block of memory:";
  for (int i = 10; i < 15; i++)
     {  cout << ptr3[i] << " ";    }
 }
 return 0;
}
```

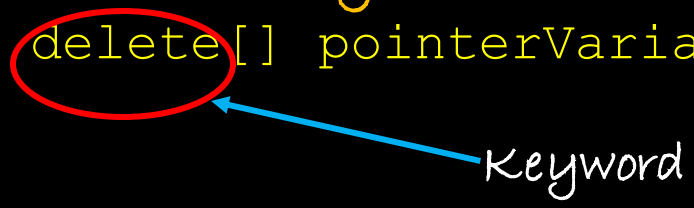
```
Pointer variable 1 : 25
Pointer variable 2 : 223.324
Value in block of memory: 11 12 13 14 15
```

The Delete Operator

- Delete is an operator in C++ used to delete the allocated memory using new operator.
- Syntax:
  - For a variable

    ```
    delete pointerVariable;
    ```
  - For an array

    ```
    delete[] pointerVariable;
    ```



Example – Delete Operator

```
class Student {
int age;
public:
// constructor initializes age to 12
Student()
{
  age = 12;
}
void getAge()
{ cout << "Age = " << age << endl; }
};
```

```
int main() {
// dynamically declare Student object
Student* ptr = new Student();
//Note: the new operator calls default constructor automatically
ptr->getAge();
// ptr memory is released
delete ptr;
 return 0;
}
```

```
Age = 12
```

Exercise 1

If an object is allocated using new operator _____

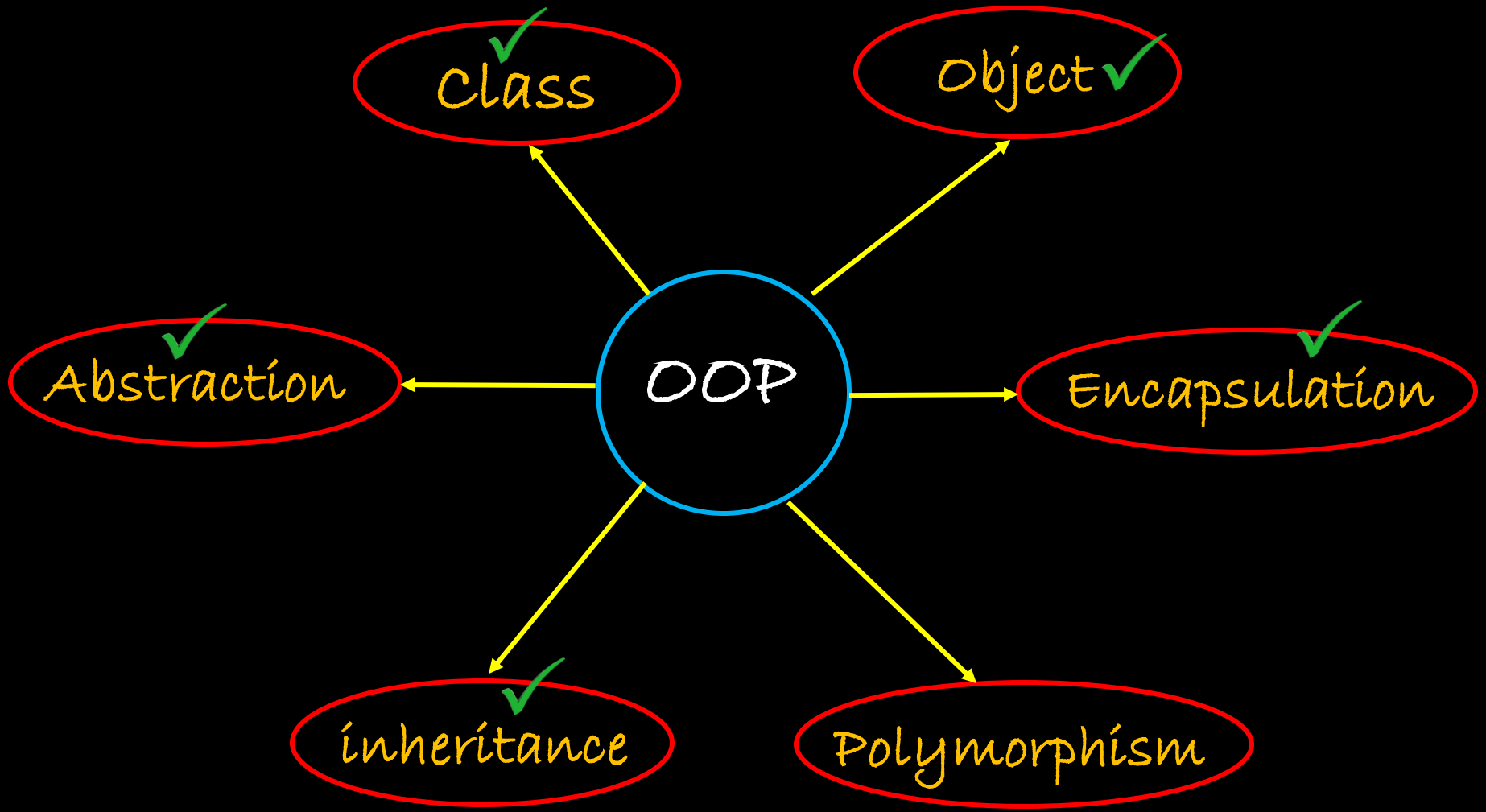
a) It should be deleted using delete operator
b) It can't be deleted using delete operator
c) It may or may not be deleted using delete operator
d) The delete operator is not applicable

- Ans: It should be deleted using delete operator

Exercise 2 – What is the output?

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

class sample {
  public:
    sample() { cout<<"Hi "; }
    ~sample() { cout<<"Bye "; }
};

int main()
{
sample *obj = new sample();
delete(obj);
return 0;
}
```

OOP overview



Quick Summary

- Static Memory
- Dynamic Memory
- Pointers
- Pointer to class
- Pointer to data member
- Pointer to member function
- New and Delete operators
- Examples
- Exercises

Up Next

Polymorphism in C++