

Object Oriented Programming (CS1143)

Week 5

Department of Computer Science

Capital University of Science and Technology (CUST)

Outline

- Static and Dynamic Arrays
- Stack and Heap
- Array as Data Members of a Class
- Array of string Objects
- Shallow Copy and Deep Copy
- Array of Objects

Arrays

- We have learnt previously how to make arrays.
- The following programs show two ways of making an array with 3 integer elements.
- Both these arrays are static as their size cannot be changed.

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      int num[3];
6      num[0]=0;
7      num[1]=1;
8      num[2]=2;
9
10     for(int i=0;i<3;i++)
11         cout<<num[i]<<endl;
12
13     return 0;
14 }
```

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      int num[3]={0,1,2};
6
7      for(int i=0;i<3;i++)
8          cout<<num[i]<<endl;
9
10     return 0;
11 }
```

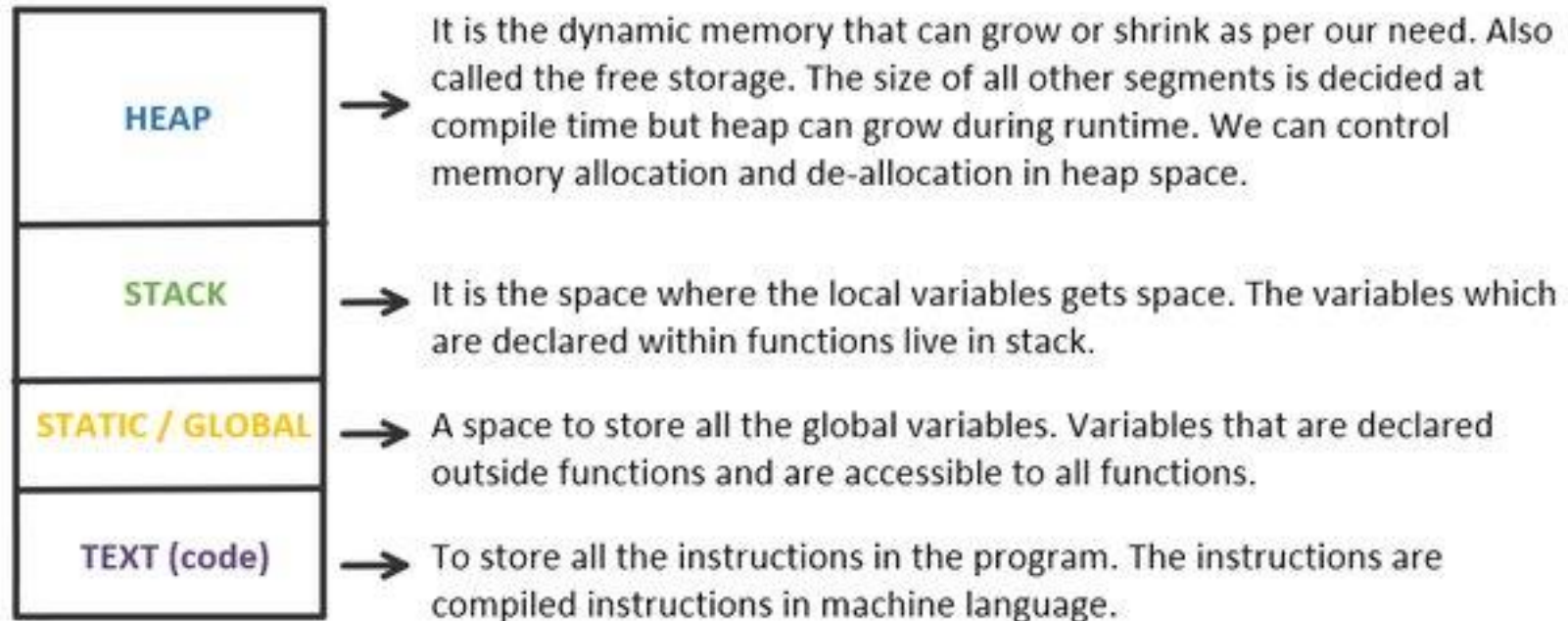
Dynamic Arrays

- In week 1, we learnt how to make arrays using dynamic memory allocation (slide 32).

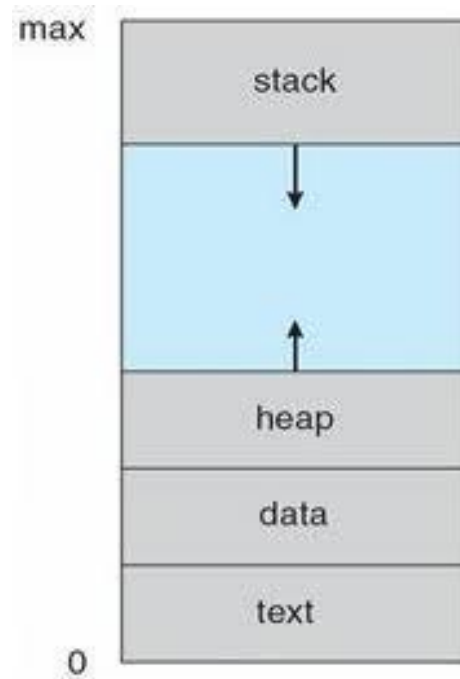
```
1  #include <iostream>
2  using namespace std;
3
4  int main ()
5  {
6      int *x=new int();
7      *x=7;
8      delete x;
9
10     int size=5;
11     int *arr = new int[size];
12     delete [] arr;
13
14     return 0;
15 }
```

Stack vs Heap

- Stack and heap both contain memory that your program can use.
- Stack is the region in the computer memory, which is automatically managed by the computer in order to store the local variables, methods and its data used by the function
- Heap is the free-floating region of memory. The memory allocated by the new operator is in this area of memory.
- The heap memory remains available until you explicitly free it or the program terminates. For example if you allocate heap memory for a variable while in a function, the memory is still available after the function returns.
- Stack and Heap grow in opposite directions (towards each other).
- Eventually, if your stack or heap grow too much, they will overlap causing an error (stack/heap overflow) and crash your program.



Heap and Stack grow towards each other



Arrays as Data Members

- Fixed size arrays are not much useful as data member since we may not know the size of the array at the time of definition.
- If we want to make arrays as data members, we must use dynamic arrays

Fixed Sized Array as Data member

```
1  #include <iostream>
2  using namespace std;
3
4  class TestArray
5  {
6      public:
7          int arr[3];
8          TestArray();
9
10 };
11
12
13 TestArray :: TestArray ()
14 {
15     arr[0]=0;
16     arr[1]=1;
17     arr[2]=2;
18 }
19
20
21 int main ( )
22 {
23     TestArray ta1;
24
25     return 0;
26 }
```

May not know this

Example: The Course Class

Description

- Data Members


- The name of the course.
- An array of students who take the course.
- The number of students (default: 0).
- The maximum number of students allowed for the course

- Member Functions

- Constructor to create a Course with the specified name and maximum number of students allowed.
- Destructor
- A function that returns the course name.
- A function that adds a new student to the course.
- A function that drops a student from the course.
- A function that returns the array of students for the course.
- A function that returns the number of students for the course.

Class Definition

```
1  #include <iostream>
2  using namespace std;
3  class Course
4  {
5      public:
6      Course(const string& courseName, int capacity);
7      ~Course();
8      string getCourseName() const;
9      void addStudent(const string& name);
10     void dropStudent(const string& name);
11     string* getStudents() const;
12     int getNumberOfStudents() const;
13
14     private:
15     string courseName;
16     string* students;
17     int numberOfStudents;
18     int capacity;
19 };
```



string* for array of strings

Member Function Definition

```
21 Course::Course(const string& courseName, int capacity)
22 {
23     numberOfStudents = 0;
24     this->courseName = courseName;
25     this->capacity = capacity;
26     students = new string[capacity];
27 }
28
29 Course::~~Course()
30 {
31     delete [] students;
32 }
33
34 string Course::getCourseName() const
35 {
36     return courseName;
37 }
38
39 void Course::addStudent(const string& name)
40 {
41     students[numberOfStudents] = name;
42     numberOfStudents++;
43 }
```


making an array dynamically
of size capacity

```
45 void Course::dropStudent(const string& name)
46 {
47 }
48
49 string* Course::getStudents() const
50 {
51     return students;
52 }
53
54 int Course::getNumberOfStudents() const
55 {
56     return numberOfStudents;
57 }
```

Main

```
61 int main()
62 {
63     Course course1("Data Structures", 10);
64     Course course2("Database Systems", 15);
65
66     course1.addStudent("Ali");
67     course1.addStudent("Ahmed");
68     course1.addStudent("Imran");
69
70     course2.addStudent("Rizwan");
71     course2.addStudent("Akeel");
72
73     cout << "Number of students in course1: " << course1.getNumberOfStudents() << "\n";
74     string* students = course1.getStudents();
75     for (int i = 0; i < course1.getNumberOfStudents(); i++)
76     cout << students[i] << ", ";
77
78     cout << "\nNumber of students in course2: " << course2.getNumberOfStudents() << "\n";
79     students = course2.getStudents();
80     for (int i = 0; i < course2.getNumberOfStudents(); i++)
81     cout << students[i] << ", ";
82
83     return 0;
84 }
```

Output

 D:\Work Umair\4_CUST (1-9-22)\1_Teaching\2_ACS1143-OOP\Practice Programs\W5-P4.exe

Number of students in course1: 3

Ali, Ahmed, Imran,

Number of students in course2: 2

Rizwan, Akeel,

Process exited after 0.05735 seconds with return value 0

Press any key to continue . . . ■

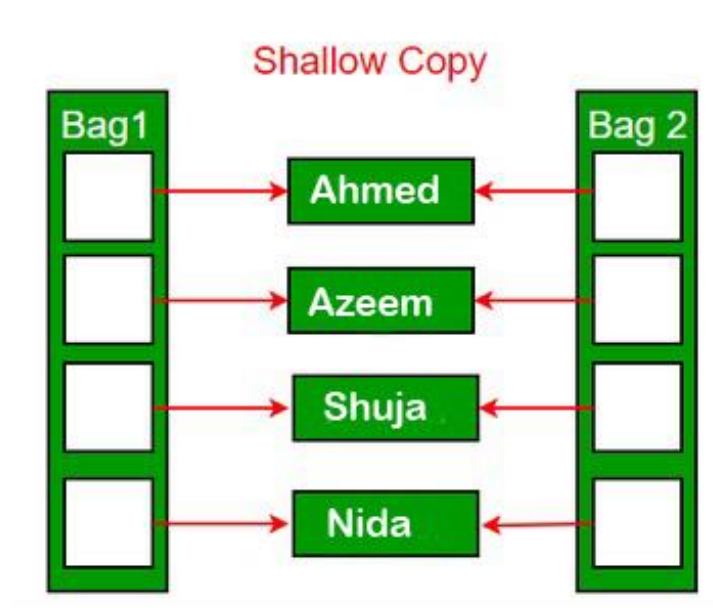
Shallow and Deep Copy

Default Copy Constructor

- We discussed copy constructor in Week 4 (slide 5)
- We also discussed that if we do not provide a copy constructor, the system provides one for us (Week 4, Slide 8).
- Such default copy constructor copies the values of one object's data members to the other.
- However if we have an array as a data member, the default copy constructor will only copy the address of one array to the other, not the values inside the array.
- This is called Shallow Copying.
- To be able to copy all the values in the array to the new object, we must write our own copy constructor. (Deep Copy)

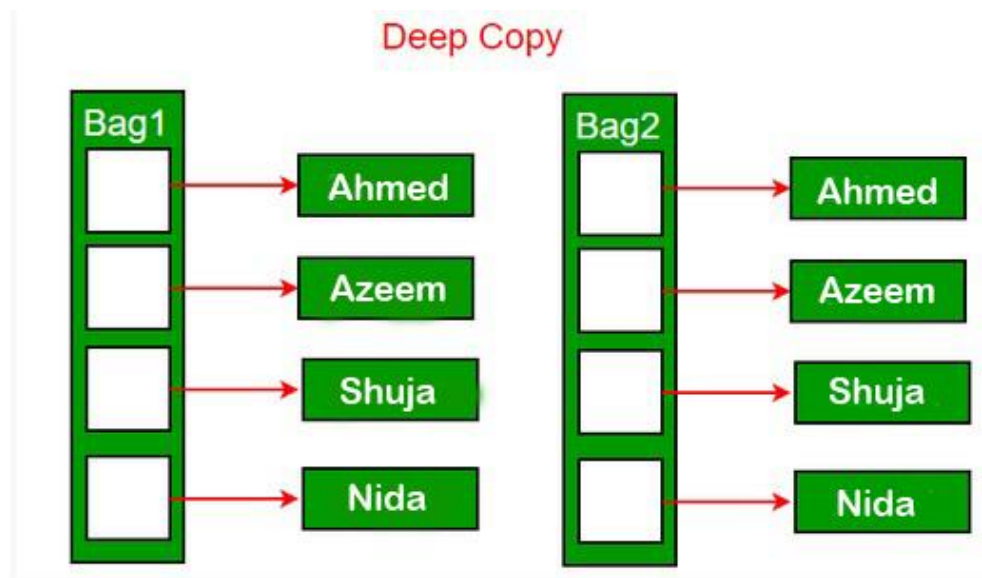
Shallow copy

In shallow copy, an object is created by simply copying the data of all variables of the original object. If some variables are dynamically allocated, then copied object variable will also reference



Deep copy

In Deep copy, an object is created by copying data of all variables and it also allocates similar memory resources with the same value to the object. In order to perform Deep copy, we need to explicitly define the copy constructor



Example: Shallow Copy

```
int main()
{
    Course course1("C++", 10);
    Course course2(course1);

    course1.addStudent("Ali"); // Add a student to course1
    course2.addStudent("Ahmed"); // Add a student to course2

    cout<<course1.getStudents()<<endl; //returns the pointer to the students array for course1
    cout<<course2.getStudents()<<endl; //returns the pointer to the students array for course2

    cout << "students in course1: " <<
    course1.getStudents()[0] << endl;
    cout << "students in course2: " <<
    course2.getStudents()[0] << endl;

    return 0;
}
```

D:\Work Umair\4_CUST (1-9-22)\1_Teaching\2_ACS1143-OOP\Practice F

0x1e6c78

0x1e6c78

students in course1: Ahmed

students in course2: Ahmed


Description

- The Course class has four data fields: `courseName` , `numberOfStudents` , `capacity` , and `students` .
- The `students` field is a pointer type.
- When `course1` is copied to `course2`, all the data fields are copied to `course2` .
- Since `students` is a pointer, its value in `course1` is copied to `course2` . Now both `students` in `course1` and `course2` point to the same array object
- We add a student "Ali" to `course1` , which is to set "Ali" in the first element of the array.
- The next line adds a student to `course2`, which is to set "Ahmed" in the first element of the array.
- This in effect replaces "Ali" with "Ahmed" in the first element of the array since both `course1` and `course2` use the same array to store student names.

Description Contd..

- When the program terminates, course1 and course2 are destroyed.
- course1 and course2 's destructors are invoked to delete the array from the heap
- Since both course1 and course2 's students pointer point to the same array, the array will be deleted twice. This will cause a runtime error.
- To avoid all these problems, you should perform a deep copy so that course1 and course2 have independent arrays to store student name
 - Write your own copy constructor and copy all the elements of the array one by one.

Runtime Error

 D:\Work Umair\4_CUST (1-9-22)\1_Teaching\2_ACS1143-OOP\Practice Programs\W5-P5.exe

0xbc6c78

0xbc6c78

students in course1: Ahmed

students in course2: Ahmed

Process exited after 2.217 seconds with return value 3221225477
Press any key to continue . . . ■



Not 0

Array of Objects

Array of Objects

- You can make an array of objects of any class
- We have already seen an array of string objects
- Now we will make an array of objects of Circle class we discussed previously

Class Definition and Member Function Definition

```
1  #include <iostream>
2  using namespace std;
3  class Circle
4  {
5      private:
6          double radius;
7      public:
8          Circle (double radius); // Parameter Constructor
9          Circle (); // Default Constructor
10 };
11
12 Circle :: Circle (double rds)
13 : radius (rds)
14 {
15 }
16 Circle :: Circle ()
17 : radius (0.0)
18 {
19 }
```

Main

```
22  int main ( )
23  {
24      //Method 1 using no parameter constructor
25      Circle c1, c2;
26      Circle arr1[2];//this will work only if you have default constructor
27      arr1[0]=c1;
28      arr1[1]=c2;
29
30      //Method 2 dynamic allocation
31      Circle c3, c4;
32      Circle* arr2 = new Circle[2];
33      arr2[0]=c3;
34      arr2[1]=c4;
35      delete []arr2;
36
37      //Method 3 using parameter constructor.
38      Circle arr3[]={Circle(2),Circle(3)};
39
40      return 0;
41  }
```

Static Variable

- Static Variable
 - A static local variable has the visibility of an automatic local variable (that is, inside the function containing it), however, its lifetime is the same as that of a global variable
 - The difference between static local variable and a global variable is that it doesn't come into existence until the first call to the function containing it
 - Static local variables are used when it's necessary for a function to remember a value when it is not being executed i.e. between calls to the function

Example

```
void counter(){
    int count=0;
    cout << count++;
}
int main(){
    for(int i=0;i<5;i++)
    {
        counter();
    }
}
```

Output : 0 0 0 0 0

```
void counter(){
    static int count=0;
    cout << count++;
}
int main(){
    for(int i=0;i<5;i++)
    {
        counter();
    }
}
```

Output : 0 1 2 3 4

Static Variable Cont..

```
#include <iostream>
float getavg (float)
Void main()
{
    float data = 1, avg;
    while( data != 0 )
    {
        cout << "Enter a number: ";
        cin >> data;
        avg = getavg(data);
        cout << "New average is " << avg <<
            endl;
    }
}
```

```
float getavg(float newdata)
{
    static float total = 0;
    static int count = 0;
    count++;
    total + = newdata;
    return total / count;
}
```

Static Member Data

- If a data item in a class is declared as static, only one such item is created for the entire class, no matter how many objects there are
- A static data item is useful when all objects of the same class must share a common item of information
 - A member variable defined as static has characteristics similar to a normal static variable: It is visible only within the class, but its lifetime is the entire program i.e. it continues to exist even if there are no objects of the class
 - However, while a normal static variable is used to retain information between calls to a function, static class member data is used to share information among the objects of a class

Static Member Data Cont..

- Why would you want to use static member data?
 - As an example, suppose an object needed to know how many other objects of its class were in the program
 - In a road-racing game, for example, a race car might want to know how many other cars are still in the race
 - In this case a static variable count could be included as a member of the class: all the objects would have access to this variable
 - It would be the same variable for all of them; they would all see the same count

Static Member Data Cont..

```
#include <iostream>
class foo
{
private:
    static int count;

public:
    foo()
    { count++; }
    int getcount()
    { return count; }
};
```

```
int foo::count = 0;
int main(){
foo f1, f2, f3;
cout << "count is " << f1.getcount() << endl;
cout << "count is " << f2.getcount() << endl;
cout << "count is " << f3.getcount() << endl;
return 0;
}
```

Output

```
count is 3
count is 3
count is 3
```

Had count not been declared static

```
count is 1
count is 1
count is 1
```

Static Member Function

- Static member functions are used to maintain a single copy of a class member function across various objects of the class
- Static member functions can be called either by itself, independent of any object, by using class name and :: (scope resolution operator)
- Syntax

```
static returnType functionName() {}
```

Function call

```
className ::functionName()
```

Static Member Function Cont..

```
class example{
private:
    static int sum;
    int x;
public:
    example(){
        sum=sum+1;
        x=sum;}
    ~example(){
        cout<<"Object Destroyed"<<endl; }
    static void exforsys() {
        cout << "\nResult is: " << sum;}
```

```
void number(){
    cout << "\nNumber is: " << x;
}
};
```

```
int example::sum=0;
void main(){
    example e1;
    example::exforsys();
    example e2,e3,e4;
    example::exforsys();
    e1.number();
    e2.number();
    e3.number();
    e4.number();
}
```

Output

```
Result is: 1
Result is: 4
Number is: 1
Number is: 2
Number is: 3
Number is: 4
Object Destroyed
Object Destroyed
Object Destroyed
Object Destroyed
```

Static Member Function Key Points

- A static member function can only access static member data, static member functions and data and functions outside the class
- You must take note not to use static member function in the same manner as non-static member function, as non-static member function can access all of the above including the static data member
- A non-static member function can be called only after instantiating the class as an object whereas static member functions can be called even when a class is not instantiated



QUESTIONS!