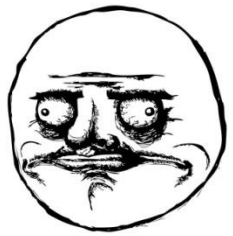
Object-oriented Programming

Lecture 9

Arrays

- In C++, we can create arrays of primitive as well as users-defined types

- These arrays can be compile-time or run-time



Syntax

- Declaring arrays at compile-time

type array_name[SIZE];

Example: *string arr[10];*

- Declaring arrays at run-time

*type * array_name;*

*array_name = **new** type[SIZE];*

Example: *string * arr;*

arr = new string[10];

Example 1

- Declaring an integer array of size 10 at compile time:

```
int main() {
    int myArr[10];
    myArr[0] = 100;
    //using array in for loop
    for(int i = 0; i < 100; i++) {
    myArr[i] = i;
    }
}
```

Example 2

- Declaring an integer array of size 10 at compile time:

```
int main() {
    int myArr[10];
    myArr[0] = 100;
    //using array in foreach loop
    for(int elem : myArr) {
    elem = 5; //sets all elements of array to 5
    }
}
```

Array of Objects

- Just like primitive data types, we can create arrays of objects

- These arrays of objects can be created at:
  - **Compile-time using default constructor**
  - **Compile-time using parameterized constructor**
  - **Runtime using default constructor**
  - **Runtime using parameterized constructor**

Compile-time Array using Default Constructor

```
class Employee {
int ID;
public:
Employee()
{ ID = 1; }
Employee(int ID)
{ this->ID = ID; }
void setID(int ID)
{ this->ID = ID; }
int getID()
{ return ID; }
};
```

```
int main() {
Employee myArr[10];
myArr[0].setID(99);

//using array in for loop
for(int i = 0; i < 10; i++)
{
cout << myArr[i].getID()
    << endl;
} }
```

Compile-time Array using Parameterized Constructor

```
class Employee {
int ID;
public:
Employee()
{ ID = 1; }
Employee(int ID)
{ this->ID = ID; }
void setID(int ID)
{ this->ID = ID; }
int getID()
{ return ID; }
};
```

```
int main() {
Employee myArr = {
    Employee(11),
    Employee(12),
    Employee(13) };

//prints 11
cout <<
    myArr[0].getID( );
}
```

Runtime Array using Default Constructor

```
class Employee {
int ID;
public:
Employee()
{ ID = 1; }
Employee(int ID)
{ this->ID = ID; }
void setID(int ID)
{ this->ID = ID; }
int getID()
{ return ID; }
};
```

```
int main() {
Employee * arr;
func(arr);
}
void func(Employee* arr)
{
    arr = new Employee[10];
    arr[0].setID(50);
    *(arr).setID(50);
    // any of the above two
    indexing syntax can be used
}
```

Runtime Array using Parameterized Constructor

```
class Employee {
int ID;
public:
Employee()
{ ID = 1; }
Employee(int ID)
{ this->ID = ID; }
void setID(int ID)
{ this->ID = ID; }
int getID()
{ return ID; }
};
```

```
int main() {
Employee * arr;
func(arr);
}
void func(Employee* arr)
{
    arr = new Employee[10];
    for(int i = 0; i < 10; i++)
    {
        arr[i] = Employee(i);
    }
    // create a new object and stores
    it in array at each iteration
}
```