Constructors and Destructors

Contents

- Scope Recap
- Constructors
  - What are they
  - Syntax
  - Default Constructors
  - Overloaded Constructors
  - Copy Constructors
  - Initializer lists
  - Resource Management
- Destructors

Canberra Institute of Technology

aie
SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX

Scope Recap

- The scope of a variable is where that variable lives, and defines its lifetime
- Variables are created when they come into scope. This can happen in a number of places
  - The start of a function.
  - Within if statements, switch statements, for and while loops.
  - Any time you define a block by adding a new set of '{ }' in your code
- Variables are destroyed when the scope they were created in is closed.
  - When a function returns
  - When control leaves an if, switch, for, or while body
  - When control leaves a block of code

Canberra Institute of Technology

aie
SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX

Constructors and Destructors.

- Constructors and Destructors are special class member functions that get called when variables are created and destroyed

- They are used for initialization and clean-up of class resources

Constructors

- Constructors are used to initialize a class object such that it is in a useable state

- Constructors are functions and can do anything a normal function can do. They typically do the following:
  - Provide default values for class variables
  - Allocate resources the class needs
  - Initialize the class variables in the class
  - Initialize variables based on arguments passed into the constructor

Constructors

- The constructor for a class is called whenever an object of that type is created

- This happens in two conditions:
  - A variable is created on the stack – the constructor is called when the variable comes into scope
  - A variable is created on the heap – the constructor is called by the new operator.

Constructor Syntax

- Declaring a Constructor is easy – make a member function with the same name as the class

```
class Player
{
public:
    Player(); //<-- Constructor
};
```

- Note that there is one large difference between constructors and regular member functions
  - Constructors have no return type

Constructor Syntax

- You define the body of the constructor just the same as you do regular member functions
  - Again, minus the return type

```
Player::Player()
{
    //Constructor body goes here
}
```

Constructor Overloading

- Constructor can take arguments, again, just like regular functions
- A class can have multiple constructors by overloading the constructor
- If a class has multiple constructors, the ‘default constructor’ is the one that takes no arguments

```
class Player
{
public:
    Player();
    Player(float my_float);
    Player(int my_int);
};
```

Using Constructors

- Seeing as we now have multiple constructors, we need a way to decide which constructor gets called
- As its name suggests, when creating a class object, by default, the default constructor is called

```
int main()
{
    //default constructor is called
    Player player_value;
    //default constructor is called
    Player * player_ptr = new Player;
    //default constructor is called on each element
    Player * player_arr = new Player[10];
}
```

Using Constructors

- To call non-default constructors, we add the same syntax as a regular function call to the end of the object declaration

```
int main()
{
    //default constructor is called
    Player player_value();
    //constructor that takes an int is called
    Player player_value(3);
    //constructor that takes a float is called
    Player player_value(6.4f);

}
```

Automatically Generated Constructors

- If you don’t make any constructors, an implicit default constructor is generated that does nothing.

```
class Player
{
public:
   int m_health;
};

//implicit empty default constructor is called
Player my_player;

//implicit default constructor is called on each element
Player player_arr[10];
```

Automatically Generated Constructors

- When you make a constructor for a class, the default constructor is no longer generated.
- This can have consequences for other areas of your code. Most notably arrays.

```
class Player
{
public:
    Player(int a_start_health);
};

//Error. No default constructor. Should be my_player(10) instead
Player my_player;

//Error. No default constructor. You cannot make an
//array without them
Player player_arr[10];
```

Constructors With Arrays

- When creating an array, the only constructor you can use is the default constructor.

- If you have defined any other constructor, you must define the default constructor – even an empty one – in order to use arrays

```
class Player
{
public:
    Player(int max_bullet_count);
    int ammo;
    int max_ammo;
};

int main()
{
    //syntax error - no default constructor
    Player player_array[100];
}
```

Using Constructors

- So what do we put in constructors, anyway?
- Constructors are used to initialize your class.
- Your goal when writing a constructor should be to put the data in your class into a useable state such as:
  - Allocating memory
  - Setting default values for variables
  - Initializing the classes contained in your class

Destructors

- As you might expect, while constructors initialize your class, destructors clean up your class.
- Destructors are called for you automatically when an object goes out of scope, or when you call delete on the object
- The syntax is also simple. Write a member function that is the name of the class, with a ~ in front.

```
class Player
{
public:
    ~Player(); //<-- destructor
};
```

Destructors

- Like constructors, destructors have no return type
- Unlike constructors they cannot take any arguments.
- Destructors pair with constructors, specifically when dealing with resource management
- The purpose of a destructor is to deallocate any resources allocated in the constructor of that class

Resource Management

- Constructors and destructors are often used to manage resources.
  - A resource is something that must be created before it can be used, and deleted after.
  - The most common resource you interact with as a programmer is memory, though there are many others
    - File handles
    - Network connections
    - Process pipes

Resource Management

- It is typically good practice to allocate all resources a class needs in its constructor and deallocate them in its destructor

- This lets you create objects on the stack and have the destructor automatically free resources without you doing it explicitly

Copy Constructor

- The Copy Constructor is a special overload of the constructor for a class

- It take in a reference to another object of the same type

```
class Player
{
public:
    Player(Player& player); //<--copy constructor
};
```

Copy Constructor

- The copy constructor is called when a class variable is declared and assigned into on the same line

```
int main()
{
    //default constructor is called
    Player p1;
    //copy constructor is called
    Player p2 = p1;
}
```

Shallow Copy vs Deep Copy

- The copy constructor is typically used to perform a deep copy of the contents of a class
- A shallow copy is simply an element by element copy of the data from one object to the next
- A deep copy is one where all data the class handles is duplicated, including the contents of references and pointers

Shallow Copy vs Deep Copy

- This is best shown with an example

- Here we see object A

Shallow Copy vs Deep Copy

- This is best shown with an example

- Here we see object A
- Object A has a pointer to an array of data

Shallow Copy vs Deep Copy

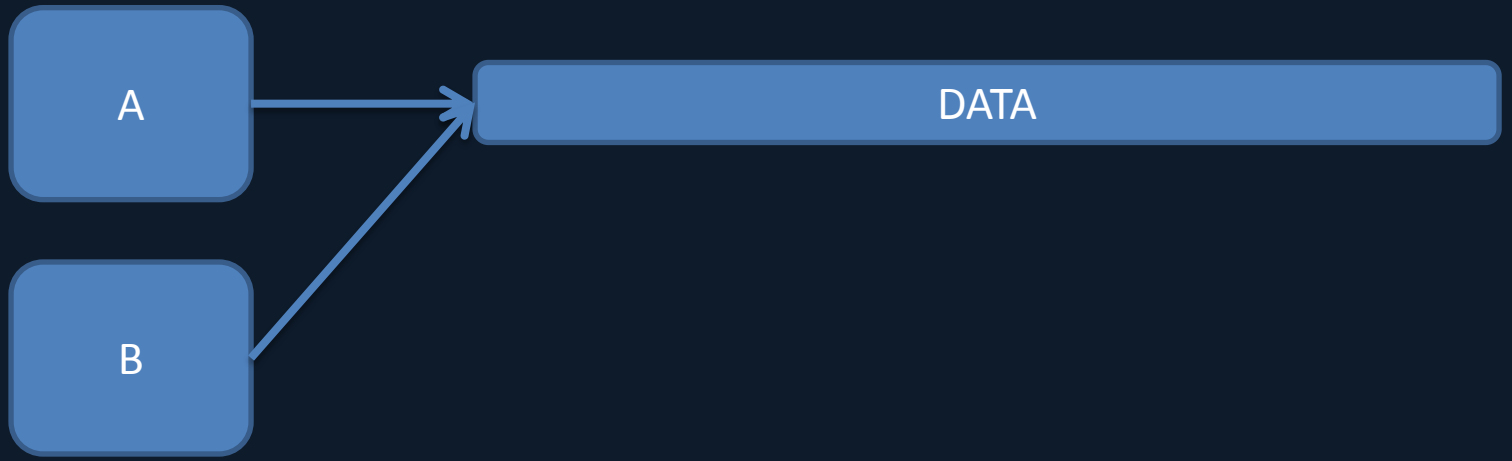
- This is best shown with an example

- Here we see object A
- Object A has a pointer to an array of data

Shallow Copy vs Deep Copy

- If we now create a second object, B and perform a shallow copy of A the pointer inside A is duplicated, so we have two pointers pointing at the same data.

- In this case, A and B share the same array



Shallow Copy vs Deep Copy

- With a deep copy, the array is duplicated as well.
- A and B now have their own identical copies of the data.

Summary

- Constructors are used to initialize your classes
- Constructors are used to set up the values within your class
- Constructors let you define how your class should be initialized
- Constructors are used to allocated resources required by your class
- Destructors are used to deallocate resources allocated in the constructor.

References

- Prata, Stephen, "*C++ Primer Plus*", 5th Ed, Chp 10, SAMS Publishing 2005

Canberra Institute of Technology

aie
SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX