



FALMOUTH  
UNIVERSITY

COMP140: Creative Computing: Codecraft

# 8: Memory

# Learning outcomes

- ▶ **Understand** Memory in modern object orientated languages
- ▶ **Compare** memory models in managed and unmanaged languages
- ▶ **Understand** the role of the profiler in measuring performance in games

# Memory



# Memory Refresher

# Memory Refresher

► Recall that:

# Memory Refresher

- ▶ Recall that:
  - ▶ Dynamic memory, allocated on the **Heap** and is **growable**

# Memory Refresher

- ▶ Recall that:
  - ▶ Dynamic memory, allocated on the **Heap** and is **growable**
  - ▶ Static memory, allocated on the **Stack** and is **fixed size**

# Stack Memory



# Stack Memory

- ▶ When you allocate value types (int, float, short, char etc), these are allocated on the stack

# Stack Memory

- ▶ When you allocate value types (int, float, short, char etc), these are allocated on the stack
- ▶ Values allocated on the stack are local, when they drop out of scope they are deallocated

# Stack Memory

- ▶ When you allocate value types (int, float, short, char etc), these are allocated on the stack
- ▶ Values allocated on the stack are local, when they drop out of scope they are deallocated
- ▶ Values passed into functions are copied onto the stack

# Stack Memory

- ▶ When you allocate value types (int, float, short, char etc), these are allocated on the stack
- ▶ Values allocated on the stack are local, when they drop out of scope they are deallocated
- ▶ Values passed into functions are copied onto the stack
- ▶ The stack is of fixed size
  - ▶ C++ Visual Studio - **1MB**

# Stack Memory Example 1

```
void Update()  
{  
    int x=10;  
    int y=10;  
  
    Vector2 pos=Vector2(x,y);  
} //<-- x, y and pos drop out of scope here
```

# Stack Memory Example 2

```
class MonsterStats
{
private:
    int health;
    int strength;
public:
    MonsterStats()
    {
        health=100;
        strength=10;
    };

    void ChangeHealth(int h)
    {
        health+=h;
    };//<- h drops out of scope here

    void ChangeStrength(int s)
    {
        strength+=s;
    };//<- s drops out of scope here
};

void main()
{
    //Create an instance of the class on the stack
    MonsterStats stats=MonsterStats();
    stats.ChangeHealth(10);
    stats.ChangeStrength(-2);
};//<-- stats drops out of scope here
```

# Heap Memory

# Heap Memory

- ▶ Otherwise known as dynamic memory



# Heap Memory

- ▶ Otherwise known as dynamic memory
- ▶ Types allocated with the **new** keyword are allocated on the heap

# Heap Memory

- ▶ Otherwise known as dynamic memory
- ▶ Types allocated with the **new** keyword are allocated on the heap
- ▶ The new operator returns a reference to the type and can be allocated to a pointer (C++)

# Heap Memory

- ▶ Otherwise known as dynamic memory
- ▶ Types allocated with the **new** keyword are allocated on the heap
- ▶ The new operator returns a reference to the type and can be allocated to a pointer (C++)
- ▶ This heap is managed by the programmer in C++ (see **delete** keyword) or the garbage collector in C#

# Heap Memory

- ▶ Otherwise known as dynamic memory
- ▶ Types allocated with the **new** keyword are allocated on the heap
- ▶ The new operator returns a reference to the type and can be allocated to a pointer (C++)
- ▶ This heap is managed by the programmer in C++ (see **delete** keyword) or the garbage collector in C#
- ▶ In C++ is very important that you delete anything allocated on the heap

# Heap Memory

- ▶ Otherwise known as dynamic memory
- ▶ Types allocated with the **new** keyword are allocated on the heap
- ▶ The new operator returns a reference to the type and can be allocated to a pointer (C++)
- ▶ This heap is managed by the programmer in C++ (see **delete** keyword) or the garbage collector in C#
- ▶ In C++ is very important that you delete anything allocated on the heap
- ▶ **for every new, you need a matching delete**

# Heap Memory

- ▶ Otherwise known as dynamic memory
- ▶ Types allocated with the **new** keyword are allocated on the heap
- ▶ The new operator returns a reference to the type and can be allocated to a pointer (C++)
- ▶ This heap is managed by the programmer in C++ (see **delete** keyword) or the garbage collector in C#
- ▶ In C++ is very important that you delete anything allocated on the heap
- ▶ **for every new, you need a matching delete**
- ▶ In the Unreal Engine objects can be Garbage Collected

# Heap Memory Example 1 - C++

```
class MonsterStats
{
private:
    int health;
    int strength;
public:
    MonsterStats()
    {
        health=100;
        strength=10;
    }

    .....
}

void main()
{
    //Create an instance of the class on the Heap
    MonsterStats * stats=new MonsterStats();
    stats->ChangeHealth(10);
    stats->ChangeStrength(-2);

    if (stats)
    {
        delete stats;
        stats=nullptr;
    }
}
```

# Passing Variable



# Passing Variable

- ▶ In C++, we can pass by value or reference. In addition to this, we can also pass in a pointer

# Passing Variable

- ▶ In C++, we can pass by value or reference. In addition to this, we can also pass in a pointer
- ▶ We can mark parameter with **&** to pass by Reference

# Passing Variable

- ▶ In C++, we can pass by value or reference. In addition to this, we can also pass in a pointer
- ▶ We can mark parameter with **&** to pass by Reference
- ▶ Custom data types and strings should be passed by Pointer or Reference

# Passing Example 1 - C++

```
int x=10;

void Adder(int &value, int v)
{
    value+=v;
}

Adder(x,10);
//x would now be 20 after this
```

# Passing Example 2 - C++

```
void SetupMonster(MonsterStats &stats, int health, int strength) ←  
{  
    stats.health=health;  
    stats.strength=strength;  
}  
  
//Calling code  
MonsterStats * goblinStats=new MonsterStats();  
SetupMonster(goblinStats,10,2);
```

# Strings



# Strings

# Strings

- ▶ When you carry out concatenations (using the `+` operator) new strings are created



# Strings

- ▶ When you carry out concatenations (using the `+` operator) new strings are created
- ▶ This means that strings can cause a performance issues

# Strings

- ▶ When you carry out concatenations (using the `+` operator) new strings are created
- ▶ This means that strings can cause a performance issues
- ▶ In C++ you should use the **stringstream** class to build strings

# Strings

- ▶ When you carry out concatenations (using the `+` operator) new strings are created
- ▶ This means that strings can cause a performance issues
- ▶ In C++ you should use the **stringstream** class to build strings
- ▶ This also allows you to create strings from other types such as int, boolean, float etc

# String Stream Example - C++

```
//We need to include the sstream header file
#include <sstream>

//Create String Stream
std::ostringstream stringStream;

//Append some text
stringStream << "Name: ";
stringStream << "Brian";
stringStream << " Health: ";
stringStream << 100;

//Grab the constructed string from string stream
std::string s=stringStream.str();
```

# Memory Management



# Memory Management in C++

# Memory Management in C++

- ▶ In C++ there is no Garbage Collection, you have to manually delete objects when no longer needed

# Memory Management in C++

- ▶ In C++ there is no Garbage Collection, you have to manually delete objects when no longer needed
- ▶ Worth repeating **for every new, you need a matching delete**



# Memory - General Good Practices

# Memory - General Good Practices

- ▶ Never allocate or deallocate during game updates

# Memory - General Good Practices

- ▶ Never allocate or deallocate during game updates
- ▶ Use Memory Pools to recycle objects

# Memory - General Good Practices

- ▶ Never allocate or deallocate during game updates
- ▶ Use Memory Pools to recycle objects
  - ▶ C++ - <http://gameprogrammingpatterns.com/object-pool.html>

# Exercise



# Exercise - All Students

- ▶ Complete exercise from last week
- ▶ Then move onto this weeks (on next slides)

# String Exercise - All Students

- ▶ Download one of the following Projects
  - ▶ BA Students - <https://github.com/Falmouth-Games-Academy/GAM160-Exercises>
  - ▶ BSc Students - <https://github.com/Falmouth-Games-Academy/COMP140-Exercises>
- ▶ Replace all string processing with StringBuilder if using C# or StringStream if using C++

# Debugging Exercise - BA Students

- ▶ Watch the following video - <https://unity3d.com/learn/tutorials/topics/scripting/debugging-unity-games-visual-studio>
- ▶ Open DebugExercise from the GAM160 Exercises Repository
- ▶ Use the debugger to find answers to the questions which are shown as comments within the Start() function



# Debugging Exercise - BSc Students

- ▶ Read the following - `https://tutorials.visualstudio.com/vs-get-started/debugging`
- ▶ Open DebugExercise project
- ▶ Use the debugger to find answers to the questions Q1 and Q2 which are shown as comments within the main function