Intro to C++ | 1.4

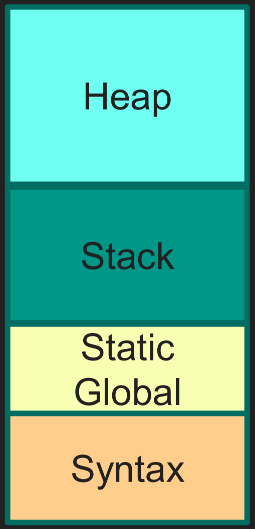
Dynamic Memory, Inheritance

Akis Zachariadis - Games Dept. Coordinator

Memory Allocation

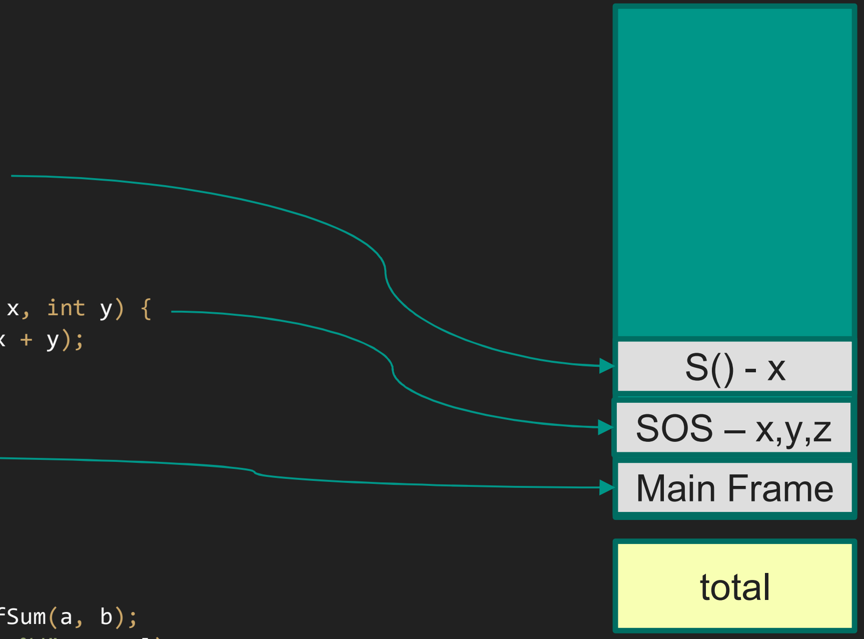
A typical program allocates its memory based on this format:

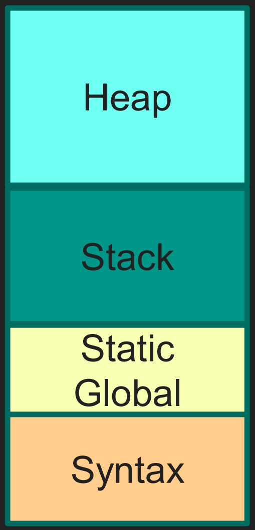
- Syntax | This is the memory that the code as text is saved
- Static / Global | This is the memory that the variables with either global or static scope are saved
- Stack | This is the memory that the program allocates on the compile process
- Heap | This is the memory that the program allocates on runtime



Stack Allocation

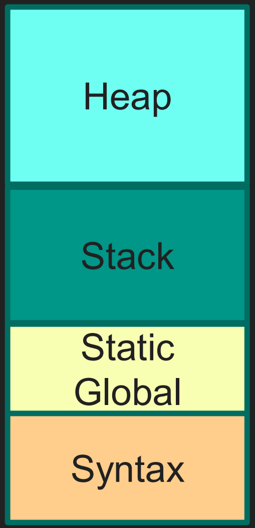
```
//Stack Allocation
#include <iostream>
#include<stdio.h>

int total;
int Square(int x) {
    return x * x;
}

int SquareofSum(int x, int y) {
    int z = Square(x + y);
    return z;
}

int main()
{
    int a = 4;
    int b = 5;

    total = SquareofSum(a, b);
    printf("output = %d", total);
}
```





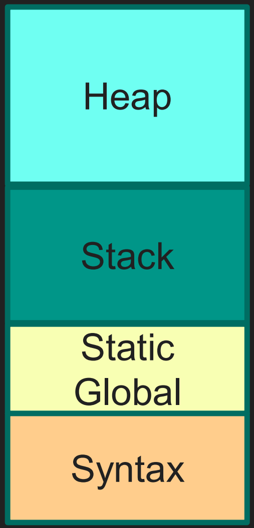
Stack Allocation

- Stack memory is for the functions and local variables
- Stack memory size is fixed and most of the times is 1 MB
- If the allocation needs more than 1MB then the program will crash. We call this situation stack overflow
- When a variable or function goes out of scope they get cleaned from the memory



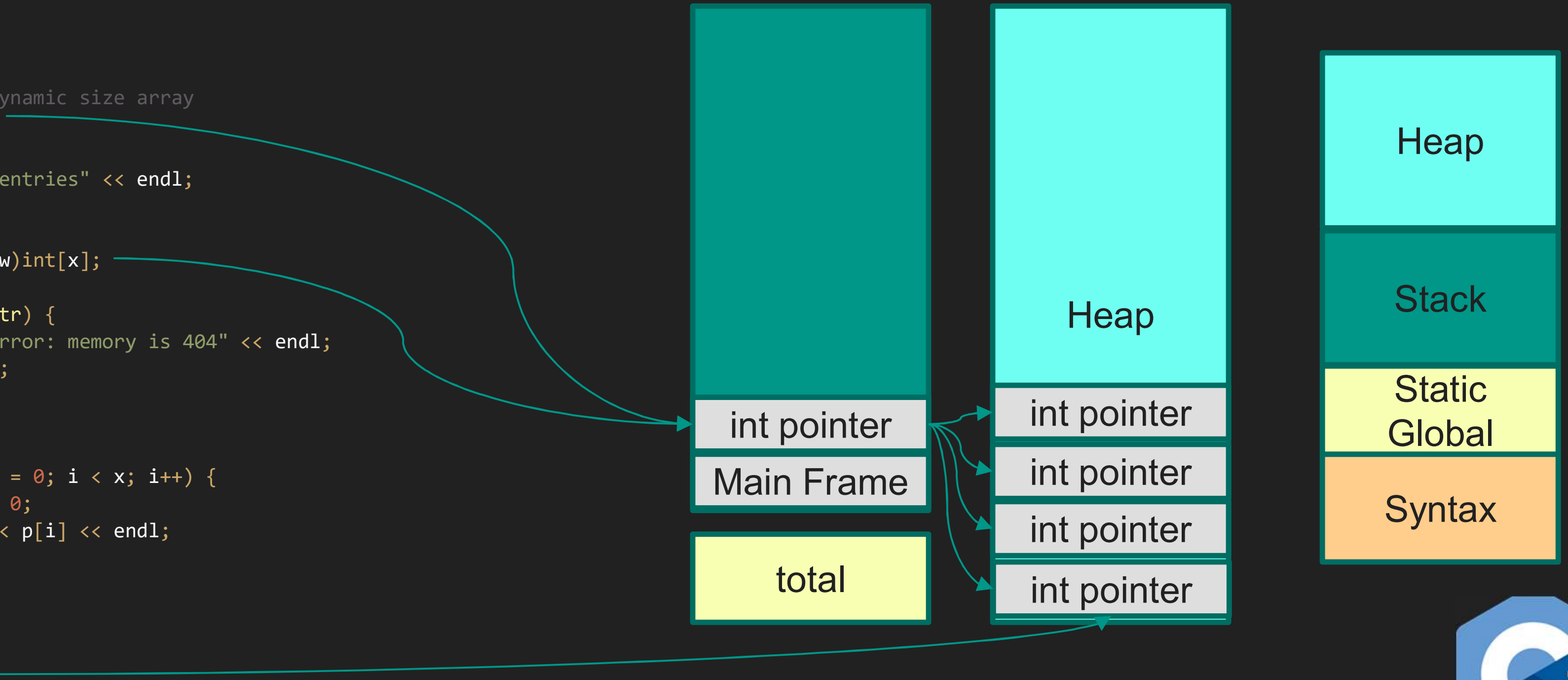
Stack Allocation

- Stack memory is an implementation of a LIFO data structure
- It works from top to bottom and it stacks on top of each other blocks of memories
- It is super fast and most of the times it fits inside a CPU cache memory. Most typical size of a CPU cache is 1-8 MB



Heap Allocation

```
//Creating a dynamic size array for the Challenge #2
#include <iostream>

using namespace std;

int main()
{
    //Creating a dynamic size array
    int x;
    int* p;
    cout << "Give entries" << endl;

    cin >> x;
    p = new(nothrow)int[x];

    if (p == nullptr) {
        cout << "Error: memory is 404" << endl;
        delete[] p;
        return 1;
    }
    else {
        for (int i = 0; i < x; i++) {
            p[i] = 0;
            cout << p[i] << endl;
        }
    }

    cin >> x;
    delete[] p;
    return 0;
}
```

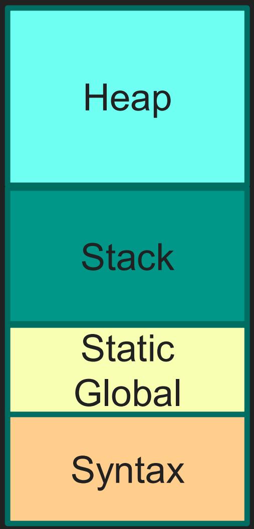


Heap Allocation

Heap memory or else free pool of memory is a dynamic size memory. The size is up to the available ram of your computer

When something goes out of scope it does not get deleted from the memory. We need to delete the memory allocation of the variable manually.

If we need to create on runtime dynamic size arrays or classes then we must use the heap memory.



Heap Allocation

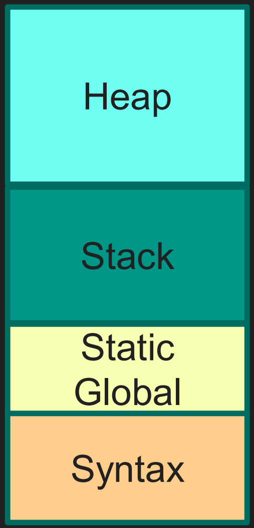
Heap memory is an unorganized memory block which is part of the RAM.

Most of the times memory addresses between two elements of an array can be in a completely different place in memory.

This can lead to CPU cache misses and to lower performance

Reasons to use the heap memory: Data does not fit in the stack memory. Game Engines use mostly the Heap Memory! A 50mb texture can not fit inside the stack memory

When we want dynamic allocation in runtime. Heavily used on Game Engines!



Header Files

- Instead of having to do forward declarations every time we can move them to a different file. This file is called a header file and the cpp file is called the source file.
- We must follow the same name convention. If we have a cpp file with the name “main.cpp” then the header file should be called “main.h”
- Then we can include the header file to the source file using the keyword include.
- With this way we can also use the header files to different source files.

Header Files Example

```cpp
//Source File
#include "Add.h"

int AddSum(int x, int y){
    return x + y;
}
```

```cpp
#pragma once

int AddSum(int x, int y);
```

Inheritance

```cpp
class Vehicle {
protected:
    string license;
    int year;
public:
    Vehicle(const string& myLicense, const int myYear)
{
        license = myLicense;
        year = myYear;
    }

    const string &GetLicense() const {
        return license;
    }

    const int GetYear() const {
        return year;
    }

    virtual const string getDescription() const {
        return license + " from " + to_string(year);
    }
};
```

```cpp
class Car : public Vehicle {
    //private variables local only to the car class
    string style;

public:
    //Constructor overload for both base class and the
child
    Car(const string& myLicense, const int myYear,
const string& myStyle) : Vehicle(myLicense, myYear),
style(myStyle) {

    }

    const string GetDescription() const {
        return license + " from " + to_string(year) +
" with style " + style;
    }

    const string& getStyle() { return style; }
};
```

Multi Inheritance

```cpp
class Truck : public Vehicle {
    //private variables local only to the car class
    string style;
public:
    //Constructor overload for both base class and the child
    Truck(const string& myLicense, const int myYear, const string& myStyle) :
Vehicle(myLicense, myYear), style(myStyle) {

    }

    const string GetDescription() const {
        return license + " from " + to_string(year) + " with style " + style;
    }

    const string& getStyle() { return style; }
};

//Multi Inheritance
class TruckCar : public Truck, public Car {

};
```

- Class TruckCar inherits from both Truck and Car
- Both Truck and Car inherits from the class Vehicle
- TruckCar implements 2 times the class Vehicle
- We can solve this but it is messy

Challenge #5

Να τροποποιηθεί το πρόβλημα του Battle Hero Simulator έτσι ώστε ο χρήστης στην αρχή να μπορεί να διαλέξει όνομα για τον κάθε ήρωα, συγκεκριμένη κλάση όπως και να επιλέξει έναν τύπο αντικειμένου. Συγκεκριμένα θα μπορεί:

Να διαλέξει για την κλάση του ήρωα ένα από τα παρακάτω:

- Warrior , Archer , Rogue, Filthy Mage

Τα στατιστικά για την κάθε κλάση θα είναι διαφορετικά.

Τα αντικείμενα που θα επιλέγει ο παίκτης θα είναι τα εξής:

- Sword ,Axe ,Stick ,Rat

Τα αντικείμενα θα προσθέτουν στατιστικά στην κλάση με την μορφή του Multiplier. Δηλαδή το stick στον filthy mage θα προσθέτει +20% damage

Στο τέλος του παιχνιδιού θα τυπωθεί το όνομα του νικητή, η κλάση που είχε όπως και το αντικείμενο του.

Additional Material

- https://en.cppreference.com/w/cpp/language/classes
- https://en.cppreference.com/w/cpp/language/object#Polymorphic_objects
- http://www.cplusplus.com/doc/tutorial/inheritance/
- http://www.cplusplus.com/doc/tutorial/polymorphism/
- https://www.w3schools.com/cpp/cpp_oop.asp
- https://www.learncpp.com/cpp-tutorial/111-introduction-to-inheritance/