717310: Game Programming

Steffan Hooper

Wednesday, 29 July 2015

Overview

- More C++
  - References
  - Pass by Value vs Pass by Reference
  - Pointers
  - Memory: The Stack

Console Game Programming

- C++ Overview: Procedural Programming:
  - References
  - Structures
  - The Stack
  - Pointers
  - Function Pointers
  - Freestore Allocations
  - Casting

Console Game Programming

- C++: References
  - Reference variables refer to the original target variable…
    - Like a pointer, without the pointer syntax.
    - Everything done to the reference, is actually done to the target…
  - Uses the **&** symbol.
  - Example:

```
int x = 47;
int& r = x;
```

References cannot be re-assigned… here **r** will always refer to **x**!!!

Console Game Programming

- C++: References continued…
  - Pass by value, parameters are copied in:

```
void func(int r)
{
    r = 10; // r is a parameter; local copy in func.
}
```

  - Pass by reference, with references:
  - Example:

```
void func(int& r)
{
    r = 10; // r refers to the caller's input var.
}
```

Console Game Programming

- C++: Structures
  - Hold data… Create complex data types…
    - Member data, fields, properties…
  - Keyword: **`struct`**
  - Example Structure Declaration:

```
struct Enemy
{
    int health;
    bool patrol;
};
```

Console Game Programming

- C++: Structures continued…
  - The **.** Operator
    - If you have a local copy of a structure, you use the **.** operator to get access to the structure's fields.

```
#include <iostream>

struct Enemy
{
    int health;
    bool patrol;
};

void passMeAnEnemy(Enemy input)
{
    input.health = 100;
    input.patrol = true;
}
```

Console Game Programming

- C++: Pass-by-Value
  - Parameters are copied…
    - Local data…

```
#include <iostream>

struct Enemy
{
    int health;
    bool patrol;
};

void passMeAnEnemy(Enemy input)
{
    input.health = 100;
    input.patrol = true;
}
```

Pass-by-Value

What is printed?

```
int main()
{
    Enemy npc;
    npc.health = 50;
    npc.patrol = false;

    passMeAnEnemy(npc);

    std::cout << npc.health;
    std::cout << std::endl;

    std::cout << npc.patrol;
    std::cout << std::endl;

    return 0;
}
```

Console Game Programming

- C++: Pass-by-Reference
  - Parameters are referred to…
    - Get to the caller's data…

```
#include <iostream>

struct Enemy
{
    int health;
    bool patrol;
};

void passMeAnEnemy(Enemy& input)
{
    input.health = 47;
    input.patrol = false;
}
```

Pass-by-Reference

What is printed?

```
int main()
{
    Enemy npc;
    npc.health = 40;
    npc.patrol = true;

    passMeAnEnemy(npc);

    std::cout << npc.health;
    std::cout << std::endl;

    std::cout << npc.patrol;
    std::cout << std::endl;

    return 0;
}
```

Console Game Programming

- C++: Pass-by-Reference
  - The **`const`** keyword can be used to ensure a variable cannot be changed.

```
#include <iostream>

struct Enemy
{
    int health;
    bool patrol;
};

void passMeAnEnemy(const Enemy& input)
{
    input.health = 47;
    input.patrol = true;
}
```

Protect the caller's data, and **avoid the copy** of the entire data structure into the **`passMeAPerson`** stack frame.

This **will not compile** as input is promised to be **`const`**, the data cannot be changed!

Console Game Programming

- C++: The Stack
  - The Call Stack:
    - An area of memory, used for storing information about the active functions used by the program.
    - At runtime:
      - Function called: Pushed onto the stack.
      - Function returns: Popped off the stack.
  - Each Stack Frame (Activation Frame) stores:
    - Parameters, Local Variables, Return Address.
    - For the called function…

Console Game Programming

- C++: Pointers
  - A variable that stores a memory address…
  - When declaring a pointer use the ***** symbol.
    - This is known as indirection.
  - addressof operator: **&**

    Beware, addressof vs reference!
    - Retrieves the address of the variable…
    - This is where the variable is stored in RAM…
  - Example:

```
int i = 47;
int* p = &i;
std::cout << i << " stored at " << p << std::endl;
```

Console Game Programming

- C++: Pointers continued…
  - Retrieve the value stored at the memory location:
    - Uses the ***** symbol.
  - Example:

```
short variableName = 72;

short* pointer = &variableName;

std::cout << pointer << " stores the value ";

std::cout << *pointer << std::endl;
```

Console Game Programming

- C++: Pointers continued…
  - Pointer terminology:
    - Null pointer:
      - A pointer with the value of zero.
    - Wild Pointer:
      - A pointer not initialised to a valid value…
    - Dangling Pointer:
      - A pointer that was once valid, but is no longer…
  - Examples:

```
int* p1 = 0;  // Null Pointer
int* p2;      // Wild Pointer
```

Console Game Programming

- C++: Pointers continued…
  - Pass by value, parameters are copied in:

```
void func(int p)
{
    p = 10; // Change the valued stored at p.
}
```

  - Pass by reference, with pointers:
  - Example:

```
void func(int* p)
{
    *p = 10; // Change the valued stored at p.
}
```

Console Game Programming

- C++: Pointers continued…
  - Beware!

```
int* func()
{
    int data = 10; // data is local to func.

    return &data; // BAD!!!
}
```

  - Do not return the address of local data!
    - Once the function returns, the stack frame is popped, and the local variables are gone!

Console Game Programming

- C++: Pointers vs Reference…
  - Which is which, what does each of the following variables store?

```
int a = 10;
int* b = &a;
int& c = a;
int d = *b;
int e = c;
int* f = &c
```

Console Game Programming

- C++: Pointers vs Reference…
  - Which is which, what does each of the following variables store?

```
int a = 10;
int* b = &a;
int& c = a;
int d = *b;
int e = c;
int* f = &c;
```

a is an integer, it stores a whole number

b is a pointer to an integer, it stores an address

c is a reference, it refers to the integer **a**

d is an integer, it stores a whole number

e is an integer, it stores a whole number

f is a pointer to an integer, it stores an address

Console Game Programming

- C++: Pointers continued…
  - The **->** Operator
    - If you have a pointer to a structure, you use the -> operator to get access to the structure's fields.

```
#include <iostream>

struct Enemy
{
    int health;
    bool patrol;
};

void passMeAnEnemy(Enemy* input)
{
    input->health = 50;
    input->patrol = true;
}
```

Console Game Programming

- C++: Pointers continued…
  - The **.** vs the **->** Operator
    - Dot is for local data's fields, arrow is to access fields via a pointer…

```
#include <iostream>

struct Enemy
{
    int health;
    bool patrol;
};

void passMeAnEnemy(Enemy input)
{
    input.health = 47;
    input.patrol = false;
}
```

Here **input** is a concrete structure…

```
#include <iostream>

struct Enemy
{
    int health;
    int patrol;
};

void passMeAnEnemy(Enemy* input)
{
    input->health = 47;
    input->patrol = false;
}
```

Here **input** is a pointer… to a struct

Console Game Programming

- C++: Pass-by-Reference
  - Using pointers…
    - Get to the caller's data…

```
#include <iostream>

struct Enemy
{
    int health;
    bool patrol;
};

void passMeAnEnemy(Enemy* input)
{
    input->health = 47;
    input->patrol = true;
}
```

Pass-by-Reference, via a pointer…

What is printed?

```
int main()
{
    Enemy npc;
    npc.health = 15;
    npc.patrol = false;

    passMeAPerson(&npc);

    std::cout << npc.health;
    std::cout << std::endl;

    std::cout << npc.patrol;
    std::cout << std::endl;

    return 0;
}
```

Console Game Programming

- C++: Stack vs Freestore Data
  - Using pointers…
    - Get to the caller's data…

```
#include <iostream>

struct Enemy
{
    int age;
    bool patrol;
};

void passMeAnEnemy(Enemy* input)
{
    input->age = 47;
    input->patrol = true;
}
```

Pass-by-Reference, via a pointer…

```
int main()
{
    Enemy local;
    Enemy* pHeap = new Enemy;
    local.age = 40;
    local.patrol = false;
    pHeap->age = 20;
    pHeap->patrol = false;

    passMeAnEnemy(&local);
    passMeAnEnemy(pHeap);

    std::cout << local.age;
    std::cout << std::endl;

    std::cout << pHeap->age;
    std::cout << std::endl;

    return 0;
}
```

What is printed?

Console Game Programming

- C++: Pointers continued…
  - Pointer Arithmetic:
    - Changing the address stored in the pointer…
      - Address change depends on pointer type…
        - » The size of the data element that is pointed to…
  - Example:

```
void func(int* pData, int numElms, int startValue)
{
   for (int k = 0; k < numElms; ++k)
   {
        *pData++ = startValue++;
   }
}
```

Console Game Programming

- C++: Pointers continued…
  - Function Pointers:
    - A variable which stores the address of a function.
  - Example:

```
void exampleIntFunc(int i)
{
    std::cout << i << std::endl;
}

int main()
{
    void (*foo)(int);
    foo = &exampleIntFunc;
    foo(32);
```

Exercises

- Recommended Readings:
  - Brownlow, M. (2004). *Game Programming Golden Rules*. Hingman, MA: Charles River Media, Inc.
  - Meyers, S. (2005). *Effective C++: 55 Specific Ways to Improve Your Programs and Designs* (3rd ed.). Reading, MA: Addison-Wesley Professional.
  - Sutter, H., & Alexandrescu, A. (2004). *C++ Coding Standards: 101 Rules, Guidelines, and Best Practices*. Upper Saddle River, NJ: Addison-Wesley Professional.

Summary

- More C++
  - References
  - Pass by Value vs Pass by Reference
  - Pointers
  - Memory: The Stack