Function Pointers

Contents

Pointers (revision)

- You have already encountered pointers in C++
  - so far you have seen how they work with variables:

```
int myVar;// An integer variable
int* myPointer; //a pointer to an integer

myPointer = &myVar; //myPointer now "points" to myVar

*myPointer = 10; //Use the pointer to change the contents of myVar;

cout << *myPointer <<endl; //use the pointer to read the value of myVar;
cout << myPointer <<endl; //We can examine the contents of the pointer

int myOtherVar;// Another integer varliabe
myPointer = &myOtherVar; //myPointer now "points" to myOtherVar
```

How are they used?

- Pointers to variables are really important in C++ and you must become proficient in their use
- They are used for many things including:
  - Passing variables into functions
  - Manipulating arrays
  - Memory management
  - Making code more generic

What are pointers?

- A pointer is a variable which contains an address in memory.
- We can use the address to “reference” variables
- We can set and read the memory which is “pointed to” by the pointer

Function Pointers

- Function pointers reference memory where code is stored
  - They point to functions
- A function pointer is just like the variable pointers you have already encountered
  - But instead of pointing to the first byte of an int or a float, it points to the first byte the function.
- Function pointers cannot be used to read or write memory. They can only be used to call functions

How is this used?

- Through the use of function pointers we can create code which is more generic. For example:
  - Modify what happens when a user presses a button on their keyboard / controller
  - Change the behaviour of enemies
  - Add call backs to optimized game play systems

How to use function pointers

- The syntax for function pointers is similar to the declaration for function prototypes
  - However instead of a function name, a pointer is created
  - When using a function pointer, the parameter of the function must agree with the function prototype it is to point to.

```
//return type (*FunctionPointerName)(parameters)
void (*Process)(float) = NULL;
```

Function pointer explained

```
//return type (*FunctionPointerName)(parameters)
void (*Process)(float) = NULL;
```

- Here, process is the name of a pointer to a function that takes a single float as an argument
  - Because functions can take and return arguments we need to provide a prototype for the function
- The pointer is initialised to NULL, but can be assigned to the address of a compatible function.

Usage – Menu System

```
bool creditsButtonPressed = false;
bool backButtonPressed = false;

//return type (*FunctionPointerName)(parameters)
void (*MenuToCall)() = NULL;

void MainMenu()
{
    if(creditsButtonPressed)
    {
        MenuToCall = &CreditsMenu;
    }
}

void CreditsMenu()
{
    if(backButtonPressed)
    {
        MenuToCall = &MainMenu;
    }
}
```

Usage – Menu System cont.

```
int main()
{
    MenuToCall = &MainMenu; //MenuToCall = MainMenu also works
    bool inGame = true;

    while(inGame)
    {
        //usual game stuff

        //Invoke the function pointer - initially calls the main menu
        MenuToCall();
    }


    system("pause");
}
```

Using typedef with function pointers

- Function pointer syntax is rather verbose
- Sensible use of typedef makes the code a bit easier to understand
- This is particularly useful if we need to write functions that return a function pointer

Usage – of *typedef* for function pointers

```
//typedef for our function prototype
typedef double(*MyFunction)(float lhs, float rhs);

//example functions
double add(float lhs, float rhs)
{
  return lhs + rhs;
}

double sub(float lhs, float rhs)
{
  return lhs - rhs;
}
```

```
//function that returns a function pointer (uses
typdef)
MyFunction getFunction(char operation)
{
  if (operation == '+')
    return &add;
  else
    return &sub;
}

//example of function usage
void functionUseExample()
{
  cout<< getFunction('+')(5, 5)<<endl;
  cout << getFunction('-')(5, 5)<< endl;
}
```

aie
SPECIALIST EDUCATORS IN
GAMES, ANIMATION & FILM VFX

Function pointers and OO

- Function pointers were fairly common in C code
- In C++ they are less common
- In OO we tend to use polymorphism to perform similar tasks
  - You may have noticed some of the examples seem a bit contrived.
- Function pointers are still very useful but they are less common in code these days.

Summary

- Revised pointers
- Introduced function pointers
- Discussed the syntax
- Discussed the used of typedef with function pointers