C++ Pointers, Memory

Readings

Pointers and Dynamic memory, stack verses heap

Intro to pointers

Excellent Reference

More Pointers

- A way to allocate/manage memory on the heap
- A way to rapidly iterate over arrays
- For C use malloc and free
- For C++ use new and delete

If up to you

- For C
  - You have to use pointers
- For C++ … Caution
  - Pointers are the source of many, many bugs
  - Use Standard Library instead, it allocates and manages heap memory for you

- malloc does not call the constructor when an object is created.
- **A* a = (A*)malloc(sizeof(A))**

- **So create using malloc, have arrays and your own data structures**
- **Or 3rd party library**

- **Why use**
- In C you don't have any support for complex datatypes such as a vector. There are also no way of passing a variable "by reference" to a function. That's where you have to use pointers. Also you can have them to point at virtually anything,

Pointers – Correct (ish)

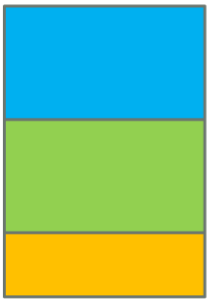
Throw an exception and things go poorly

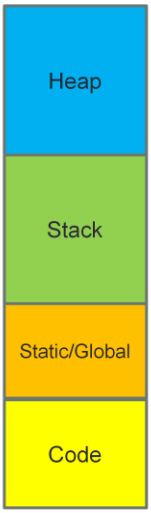
```
const int MY_SIZE = 10;

bool dynamic_good() {
    int *p = new int[MY_SIZE];

    //do some work

    //free if allocated
    if (p)
    {
        delete[] p;
        p = 0;
    }
}
int main()
{
    dynamic_good();
    return 0;
}
```





- //<u>dont forget to free it,</u>
- <u>not</u>//very robust though,
- if exception is thrown
- //you never free this memory, better to use
- //objects to allocate and free memory (coming soon)
- //or try-catch (coming soon)
- //

- int number; int *pNumber = number; delete pNumber; // wrong - *pNumber wasn't allocated using new.or on some compilers(MSVC) __try __finally (coming maybe)

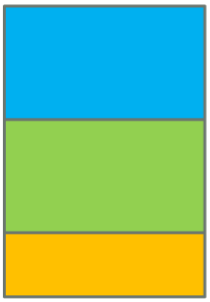
Pointers - Dereference

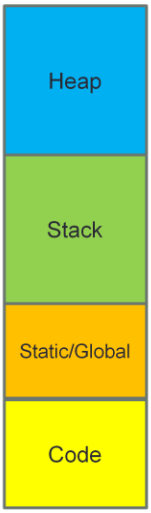
```
void pointerDereference(){
    int     *pInt   = 0;
    int     *pInt2  = 0;
    int myInt[5]    = {0,1,2,3,4};

    //2 ways to set pointers
    //to an array
    pInt        = &myInt[0];
    pInt        = myInt;

    pInt2 = pInt;

    for (int i=0;i<5;i++)
    {
        cout<< *(pInt + i) <<" ";
    }
    cout<<std::endl;
}
```





- **char* a = "Hello";**
- **char a[] = "Hello";**

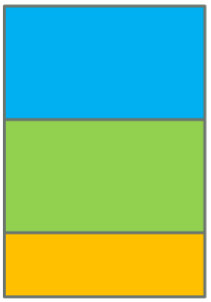
Pointers – Dangling

```
int *p = new int[MY_SIZE];
if (!p)
    return false;

int *p2 = p;

if (p)
{
    delete [] p;
    p=0;
}

//what about p2?
```



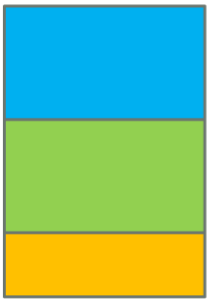
- **Better not try to dereference it**
- **Should set to 0**

Pointers – Memory Leak

```
const int MY_SIZE =10;

bool dynamic_memleak() {
    int *p = new int[MY_SIZE];
    if (!p)
        return false;

    return 0;
}
```



- **//p is a local <u>var on the stack when this function exits it disappears</u>**
- **//along with only way to free the MY_SIZE <u>ints we just allocated</u>**
- **//the new new <u>int[MY_SIZE] allocates 10 ints on the heap p points to them</u>**

Passing Pointers - review

```
char myString[] = "I am at an alpha low";
char *pChar = myString;

pointerByValue(pChar);
pointerByRef(pChar);
```

```
//pointers by value
void pointerByValue(char *myPointer){

//pointersbtyRef
void pointerByRef(char *&myPointer){
```

- **1st can dereference what pointer is pointing to only**
- **2nd all that and can change pointer**

Pointers - different types

- Pointers to different types are different
- Cannot (for the most part) assign 1 to another

```
int *pInt =0;
double *pdouble = 0;
pInt = pdouble;
```

Syntax Error

- **Can assign pint=pChar though**

Pointers and const

```
//trick mentally draw a vertical line thru pointer asterix
//const to left     -whats pointed to is constant
//const to right    -pointer itself is constant

char                    *p1 = "hello";  //non const pointer
                                        //non const data
const   char            *p2 = "hello";  //non const pointer
                                        //const data
char    *       const   p3 = "hello";   //const pointer
                                        //non const data
const   char*   const   p4 = "hello";   //const pointer
                                        //const data
```

Pointer tip

- If you create something using new[]
- You must delete using delete[]

- If you create something using new
- You must delete using delete

- **//Example**

```
int *p=new int[10];
delete p; //undefined behaviour, sometimes OK sometimes
          //not
```

- **There's a common myth among programmers that it's OK to use delete instead of delete [] to release arrays built-in types.**

- **This is totally wrong. The C++ standard specifically says that using delete to release dynamically allocated arrays of any type yields undefined behavior. The fact that on some platforms, applications that use delete instead of delete [] don't crash can be attributed to sheer luck:**
- **Furthermore, there's no guarantees that this code will work on other compilers.**

Conclusions

- Pointers are dangerous
- Please study this lecture, readings (especially intro one) and the example programs online
- We will see more pointers as we start on objects.

–Show pointer project