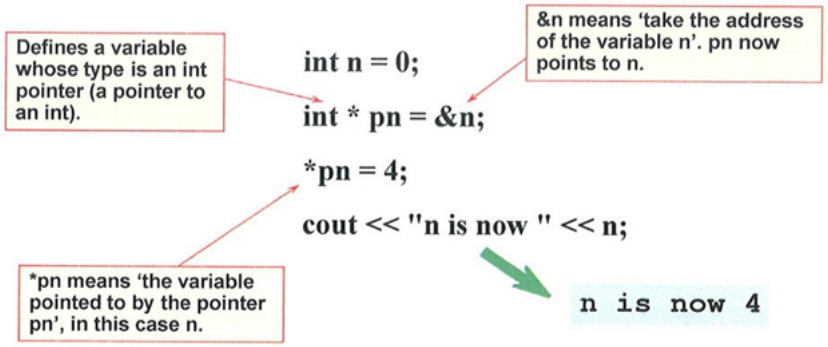
Pointers

Pointers

- A pointer is a variable that holds the address of another item
  - Variable
  - Function
  - Another pointer(!)
- Allows for extremely efficient implementations of code
  - Can also be very dangerous
- Introduce new syntax
  - & operator: return address of an item
  - * operator: return the item pointed to by this pointer
  - * also used in declarations to specify a pointer

Pointers Example

- Using a pointer to manipulate an int variable



Pointers and Dynamic Objects

- Pointers allow us to refer to objects (variables, arrays, etc) that are created during program execution
  - Dynamic objects
- Gives considerably more flexibility to programs in using memory more efficiently
  - E.g. array to store a set of values unknown at compile time
- This comes at a cost in complexity
  - Lifetime of dynamically created objects is also down to the program
  - Most C++ (and C) problems are related in some way to dynamic objects

A Simple Example

- Creating and manipulating a double variable dynamically

```
double * pNum = new double;
```
new creates the dynamic item and returns a pointer - its location in memory

```
*pNum = 7.5;
```
The item 'pointed to' is accessed using *

```
cout << "Number is " << *pNum << endl;

delete pNum;
```
The dynamic item is deleted when no longer required

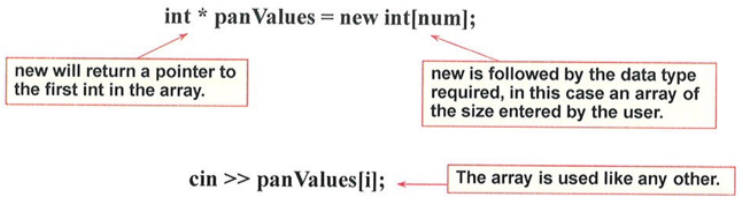
```
pNum = 0;
```
The pointer is set to 0 to prevent the memory location holding the deleted item being used or deleted again. From C++11 can also use nullptr rather than 0

Dynamic Arrays

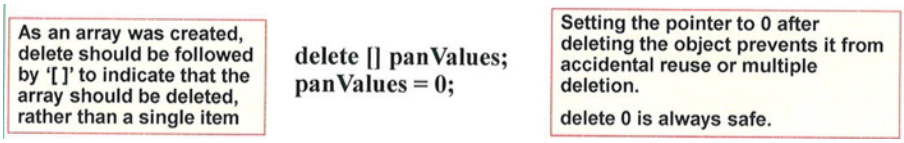
```
cout << "How many items?: ";
size_t num;
cin >> num;
int * panValues = new int[num];
for ( size_t i = 0; i < num; i++ )
{
        cout << "Enter value " << i << ": ";
        cin >> panValues[i];
}
int sum = 0;
for ( size_t i = 0; i< num; i++ )
        sum += panValues[i];
cout << "Sum is " << sum << ", average is " << sum / num << endl;
delete [] panValues;
panValues = 0;
```

Dynamic Arrays

- The new operator is used with the required number of elements to create the array



- When no longer required, the object must be deleted



Pointer Arithmetic

- Pointer variables support limited arithmetic
  - Add/subtract
  - Result is a new address in memory
- Result is scaled by the size of the object pointed to

```
int i = 10;
int j = 12;
int *ip = &i;    // ip points to I
ip++;            // ip points to the next int which happens to be j
```

Pointer Arithmetic and Arrays

- Pointer arithmetic gives an alternative way of working with arrays:

```
int * panValues = new int[num];
for ( size_t i = 0; i < num; i++ )
{
        ...
        cin >> *( panValues + i );
```

The loop counter i is added to the pointer. Note that this adds one whole object to the pointer, not one byte. In this case it means that it points to the next int.

C Style Strings

- Traditional way to represent strings in C and early C++
  - Array of char, one element for each character in the string
  - One extra byte, containing 0 numeric after the last character
  - Allows efficient manipulation of strings

pszFirst points to an array of chars containing the 5 letters from 'Hello' followed by a binary 0.

```
const char * pszFirst = "Hello";

char achArray1[5] = "C++!";

char achArray2[] = "I can't be bothered to count";
```

char arrays can be initialized to the contents of a string. If the array size is not provided it is calculated automatically. If you specify the size, remember the terminating binary 0.

How Long is a String?

- Library function strlen() can be used to determine length of a C style string
  - Don't rely on sizeof operator

```
const char * pszFirst = "Hello";
char achArray1[10] = "C++!";
cout << strlen( pszFirst ) << " , " << sizeof( pszFirst ) << endl;
cout << strlen( achArray1 ) << " , " << sizeof( achArray1 ) << endl;
```

strlen gives length of the strings themselves, not including the terminating 0s.

```
5 , 4
4 , 10
```

sizeof(pszFirst) gives 4, the number of bytes in the pointer, not the string pointed to.

sizeof(achArray1) gives 10, the number of bytes in the whole array, not just the part filled by the string.

Strings in Modern C++

- Standard library has definition of string type as a class
  - Nowadays prefer this to C style arrays where possible

```
char ach1[10], ach2[10];
string s1, s2;
cin >> ach1;
cin >> ach2;
cin >> s1;
cin >> s2;
```

```
hello
hello
hello
hello
Different
Same
```

```
if ( ach1 == ach2 ) cout << "Same\n"; else cout << "Different\n";
if ( s1 == s2 ) cout << "Same\n"; else cout << "Different\n";
```

ach1 == ach2 compares the pointers to the character arrays.
s1 == s2 actually compares the strings.

Combining Strings

- Using C style implementation
  - Beware memory problems!!

```
const char * psz1 = "Original";
const char * psz2 = "More";
char * pszNew = new char[strlen(psz1) + strlen(psz2) + 1];
strcpy( pszNew, psz1 );
strcat( pszNew, psz2 );
```

Create a new char array big enough for both strings and a terminating binary 0.

strcpy copies the original string into the new array. strcat concatenates the 2nd string onto the end of the first. Note that the parameters for these functions are destination, source.

Combining Strings

- Using string class

```
string s1 = "Original";
string s2 = "More";
s1 += s2;
```

s1 now contains the original contents of s1 followed by s2.

- Notice not char pointers!!

```
char * psz1 = "Original";
char * psz2 = "More";
psz1 += psz2;
```

Only an integer can be added to a pointer. This attempt to add a pointer to a pointer is rejected by the compiler, but even if accepted would result in psz1 containing the memory address of the first string added to the memory address of the 2nd string, which would be the address of an arbitrary memory location.