COMSC-165 Lecture Topic 9
Intro to C Pointers

Reference
Deitel, chapter 7
Tutorial

Memory Addresses
also known as "memory locations" or "locations"
variable and objects have:
  names *(who)*
  data types *(what)* 1, 2, 4,... bytes
  values *(how much)*
  memory locations *(where)* 4 bytes
`int i;` -- value is `i`
`int i;` -- memory address is `&i`
  symbol: the leading ampersand
`cout << i << ' ' << &i;`

Pointer: a Variable that Stores a Memory Address
`int* p;` can store `&i`'s value: `p = &i;`
`int* a = new int[n];` stores address of allocated array memory

Pointers: Old Syntax for Pass By Reference
C had *pointers*; C++ has *pass by reference*
new syntax, using pass-by-reference
  symbol: the trailing ampersand
```
setToNoon(someTime);
void setToNoon(tod& t)
{
  t.hour = 12;
  t.minute = 0;
  t.second = 0;
}
```

old syntax, using pointers
  `setToNoon(&someTime);`
  symbol: the trailing asterisk
```
void setToNoon(tod* t)
{
  t->hour = 12; // arrow notation
  t->minute = 0; // arrow notation
  t->second = 0; // arrow notation
}
```

the "thing pointed to by" operator...
symbol: the leading asterisk (or "star")
```
getAvg(..., &avg, ...);
*avg = sum / n;
```
pointer declaration syntax options:
```
int* p; // instructor's way
int *p; // int a, b, *p;
```

Pointer Arithmetic
pointer incrementing thru an array
  `*p++` *(not possible w/ref's)*
  that's why it needs a type!
array's name is a pointer
```
int nums[10]; // an array
int* p; // a pointer to an int
p = nums; // point to zeroth int
```
  string name *is a pointer*
  `char s[] = "Hello";` // "s" is alias for array's memory address
  `cout << strlen(s);` *(not `&s`)*

Pointer Arrays For Indexing
an index to an array
  alphabetize `theTime[5]`
  order by time `theTime[5]`
  `tod& index[5]` ???
    array of refs *not allowed!*
  `tod* index[5];`
    array of pointers is valid
coded example of building an index
  `theTime[5]`
the concept of Linked Lists
  use pointers as struct data members
  ref's cannot be changed; pointers can!

Clever Pointer Uses
strcpy
```
while(*d++ = *s++);
```
strlen
```
int i = 0;
while(*s++) i++;
return i;
```
strcmp
```
while(*d)
  if(*d++ != *s++) return *--d - *--s;
return -*s;
```

`const` Pointers
pointers can access TWO values:
  their own value: a memory address
  the value stored at that address
so `const` needs to have TWO meanings:
  protect the memory address that it stored
  protect the value at that address
so here's how:
  `int* const p` protects the memory address
  `const int* p` protects the value at that address
  `const int* const p` protects *both*
refer to these as "leading" and "trailing" consts

```
int * p; // okay, too
```

dereferencing equals aliasing
if `int i; int* p = &i;`, then
`*p` is alias for `i`

Pointer Values

three possible values

1. "valid"
   memory address of an actual, current variable
2. "wild"
   memory address of nothing
   uninitialized,
   or address of former variable
3. "not in use"
   NULL or 0 (zero)
   NULL requires *any* #include

Advanced Uses Of Pointers

to store locations of functions
"pointers to functions"
e.g., `void (*)(tod&)`
to store locations of other pointers
"pointers to pointers"
e.g., `tod** p;`

Array Vs. Pointer Notation

`x[0]` same as **`*x`**
`x[1]` same as **`*(x + 1)`**
`&x[i]` same as **`x + i`**
`void fun(int* x)`
same as **`void fun(int x[])`**

Pointers

1. A **pointer** is another variable that stores a whole number, like int or long or unsigned int or short int, etc.
2. Pointer **values** are memory locations -- where in memory that *another* variable is stored.
3. Pointer **data types** track the *type* of variable stored in that location.

```
int x; // a regular variable stored at some location in memory
int* p; // a variable, capable of storing a memory location -- this "variable" is called a "pointer"
p = &x; // & gets the memory location where x is stored, = copies it to the pointer
... *p ... // is an ALIAS for x, since p "points to" x

int y;
p = &y;
... *p ... // is NOW an ALIAS for y, since p "points to" y

tod t; // an "object" variable
tod* p = &t; // a pointer, storing the location of the object "t"
... *p ... // is an alias for t
... ~~*p.hour~~ ... // does NOT work because . gets done first, then *
... (*p).hour ... // solves the problem
... p->hour ... // is shorthand for the above
```

```
// How to read the filenames from the command
//  line rather than from the user prompts.
//
// e.g., A:\>iocopy a:\test.txt a:\testcopy.txt
//
// In VisualC++, use this to enter command line args:
// Project->Settings->Debug->Program Arguments

#include <iostream>
...
#include <cstring>

int main(int argn, char** args)
{
  char infile[256]; // a C string
  char outfile[256]; // another C string
  if (argn == 3)
  {
    strcpy(infile, args[1]); // strcpy is in...
    strcpy(outfile, args[2]); // ...the cstring library
  }
  else
  {
    cout << "Enter the file to be COPIED: ";
    cin.getline(infile, sizeof(infile));

    cout << "Enter the file to be CREATED: ";
    cin.getline(outfile, sizeof(outfile));
  }

  ...
}
```