Lecture 12 Notes

Derek Harter

2015-04-21

1 First Session (11 - 11:40)

1.1 Relationship Between Pointers and Arrays

- Arrays and pointers are intimately related in C++ and may be used (almost) interchangeably.
- An array name can be thought of as a constant pointer.
- Pointers can be used to do any operation involving array sub scripting (using pointer arithmetic, see next).

```
int b[5]; // create 5-element array of integers
int* bPtr;
```

- Because the array name (without a subscript) is a (constant) pointer to the first element of the array, we can set bPtr to the address of the first element in array b

```
bPtr = b;
```

- equivalent to assignment the address of first element

```
bPtr = &b[0];
```

1.2 Pointer Arithmetic

- Pointers are valid operands in arithmetic expressions. (e.g. +/-)
- Not all operators used in these expressions are valid with pointer variables (/? %?)

- Can increment or decrement a pointer. Basically like iterating through index of an array.
- Can add or subtract to a pointer, and can subtract one pointer from another.
- Pointer arithmetic is usually only sensible in context of processing arrays in memory.
- For array `b` and `bPtr`, equivalent ways to access the third element

```
b[3];
*(bPtr + 3);
```

- The 3 is the offset to the pointer. Both expressions calculate the address that is 3 (integers) past where `b` / `bPtr` are pointing, and dereference that value.
- The address of the 3rd element, in eqivalent notation

```
&b[3];
bPtr + 3;
```

2 Second Session (11:45 - 12:30)

2.1 Character Arrays and Pointer-Based String Processing

- Old style (legacy) string processing
- Should probably prefer to use string type whenever possible for new problems
- However, helps to understand arrays and pointers, and helps if you have old code using old c-style strings
- A string literal "blue" can be assigned to a string type, and it can be assigned to initialize an array of char for old-style C string processing.
- If you don't specify array size, the size allocated is inferred from the string literal.

```
char color[] = "blue";
const char* colorPtr = "blue";
```

- each initialize a variable to a string "blue".
- Since arrays of characters and a pointer to the beginning of such an array can be used interchangeably, you can do either to create an array of characters.
- old style C-strings use an implicit flag character, the null character ~"~ as string terminator.
- For use when passing strings to old-style string processing functions, like `strlen` and `strcpy`.
- NOTE: compare this method to method you have been taught for passing arrays to functions.

3 Third Session (12:40 - 1:40)

3.1 Arrays of Pointers

- Arrays may contain pointers (e.g. can have an array of pointers).
- Useful for dynamically sized structures, or efficiency reasons.
- A common usage is an array of char*, e.g. an array of old c-style strings

```
char* colors[] = {"red", "orange", "yellow", "green", "blue", "indigo", "violet" };
```

3.2 Command Line Arguments

- `main()` function is a **function**. Takes parameters as input and returns result.
- command line arguments given by user at a terminal when execute program.
- Passed in through the `argc` and `argv` parameters to `main`.
- `argv` is an array of `char*`. Each pointer points to a string of a parsed command line argument.

```
int main(int argc, char* argv[])
```

3.3 Function Pointers

- A pointer to a function contains a function's address in memory.
- Can pass pointers to functions as parameters to other functions.
- Can return function pointers as results from other functions.
- A useful technique, used extensively in functional programming.
- For example, can specify a order function for sorting routines, so can change sorting order (ascending, descending, sort by different keys, etc.) without changing logic of the sort function.
- Used for example in the C standard libraries.