Review from Thursday’s Lecture — C++ Classes; Pointers
Koenig and Moo: Sections 10.1-10.4

- Example of more sophisticated struct and a map
- Rewriting the week 4 homework solution to use classes. Result is now posted on the course web site.
- Pointers
- Arrays

Today’s Lecture — Pointers and Dynamic Memory

Rest of Chapter 10

- Arrays and pointers
- Different types of memory
- Dynamic allocation of arrays
- Examples and applications
- Command line arguments

Three Types of Memory

- Automatic memory: memory allocation inside a function when you create a variable. For example:

```c
int x;
do uble y;
```

This allocates space for local variables in functions and eliminates it when variables go out of scope.

- Static memory: variables allocated statically, as in
aren't eliminated when they go out of scope. They retain their values, but are only accessible within the scope where they are defined. We will not be discussing these much.

- Dynamic memory: explicitly allocated as needed. This is the focus of today’s lecture.

Dynamic Memory

- Dynamic memory is
  - created using the new operator,
  - accessed through pointers, and
  - removed through the delete operator.

- Here’s a simple example involving dynamic allocation of integers:

```cpp
int * p = new int;
*p = 17;
cout << *p << endl;
int * q;
q = new int;
*q = *p;
*p = 27;
cout << *p << " " << *q << endl;
int * temp = q;
q = p;
p = temp;
cout << *p << " " << *q << endl;
delete p;
delete q;
```

- The expression new int asks the system for a new chunk of memory that is large enough to hold an integer.

- The statement delete p; takes the integer memory pointed by p and returns it to the system for re-use.

- Inbetween the new and delete, the memory is treated just like memory for an ordinary variable, except the only way to access it is through pointers.
• Hence, the manipulation of pointer variable and values is similar to the examples covered in Thursday’s lecture notes except that there is no explicitly named variable other than the pointer variable.

• Dynamic allocation of primitives like ints and doubles is not very significant, however. What’s more important is dynamic allocation of arrays.

Dynamic Allocation of Arrays

• Declaring the size of an array at compile time doesn’t offer much flexibility.

• We can dynamically allocate an array based on data. This gets us part-way toward the behavior of a vector.

• Here’s an example:

```c++
int main()
{
    cout << "Enter the size of the array: ";
    int n;
    cin >> n;
    double *a = new double[ n ];

    int i;
    for ( i=0; i<n; ++i )
        a[i] = sqrt( i );

    for ( i=0; i<n; ++i )
        if ( double(int(a[i])) == a[i] )
            cout << i << " is a perfect square " << endl;

    delete [] a;
    return 0;
}
```

• Consider the line

```c++
double *a = new double[ n ];
```
The expression `new double[ n ]` asks the system to *dynamically* allocates enough consecutive memory to hold `n` double’s (usually `8n` bytes).

* What's crucially important is that `n` is a variable.
* Therefore, its value and as a result the size of the array is not known until the program is executed.
* When this happens, the memory must be allocated dynamically.

- The address of the start of the allocated memory is assigned to the pointer variable `a`.

- After this, `a` is treated like it is an array. For example,

  ```
  a[i] = sqrt( i );
  ```

- In fact, the expression `a[i]` is exactly equivalent to the pointer arithmetic and dereferencing expression

  ```
  *(a+i)
  ```

  which we have seen several times before.

- After we are done using the array, the line

  ```
  delete [] a;
  ```

  releases the memory allocated for the entire array. Without the `[]`, only the first double would be released.

- Since the program is ending, releasing the memory is not a major concern. In more substantial programs it is ABSOLUTELY CRUCIAL.

**Dynamic Allocation: Arrays of Class Objects**

- We can dynamically allocate arrays of structs and class objects:

  ```
  int n;
  cin >> n;
  struct foo {
    string a, b;
  };
  foo * farray = new foo[n];
  ```
• For class objects, the default constructor (the constructor that takes no arguments) must be defined. For example, in the Course we wrote last week, there is no default constructor.

  – Fortunately, vectors do not require default constructors — another advantage of vectors over arrays.

Dynamic Allocation Example: The Sieve of Eratosthenes, Revisited

We return to the problem of finding all primes. The first time we did this we used a list. Now we use a dynamically allocated array. The code is in all_primes_array.cpp.

• I have slipped a new mechanism in here: command-line arguments. We’ll discuss this in a few minutes.

• Once $n$ is taken from the command-line, the lines

  bool * is_prime_sieve;
  is_prime_sieve = new bool[ n+ 1 ];

define a bool pointer and then make it point to the start of a dynamically allocation array of bools.

  – Each entry from 0 to $n$ is used to indicate whether or not the index value is prime.

• After this, is_prime_sieve is treated like an ordinary array.

• The dynamic memory is deleted at the end.

For comparison purposes, a third version of the Sieve of Eratosthenes program is also provided. It is logically very similar to the second version but uses vectors instead of arrays.

Command-Line Arguments

• We run this program from the command line as follows:

  primes.exe 100

  in order to find the primes less than or equal to 100.
• Associated with this, the line

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

is a different declaration of the main function.

• The value argc is the number of strings (yes, strings!) on the command line. In this case there are two:

        primes.exe
        100

• These strings are stored in argv, which is an array of pointers to C-style strings.

• Sometimes the definition is written

        int main( int argc, char** argv )

indicating, equivalently, that argv is a pointer to a char pointer.

• The line

        int n = atoi( argv[1] );

converts the string "100" in argv[1] to the integer 100 and stores it in n.

        – atoi is an abbreviation for “ascii to integer”

• The prototype for function atoi is in cstdlib.

• There is a similar function atof to convert a string to a float (or double).

Looking Ahead to Lab

The lab on Wednesday will explore a bit on dynamic memory allocation, and more on command-line arguments. It will also explore file input and output, which we have not yet covered in class. You can (and should) read about this in Section 10.5 of the text.
Looking Ahead to Thursday and the Week After Break

We will diverge a bit from the order of coverage of the material. We will talk about classes and operators, which is covered in Chapter 12 of the text, but our discussion will be based first on a simpler example. In looking at the string example, class Str, in Chapter 12, think of the Vec container as a vector.