Lecture 15

- Function pointers cont.
- Unions
- Assignment operator
- Class example

Function Pointer Example (2)

- Library function **qsort** ("Quicksort")
- Generic sorting routine
- Average time complexity ~ C * n * log n
- Worst case time complexity ~ C' * n * n
- man qsort :

```
#include <cstdlib>

void qsort(void *base, size_t nmemb, size_t size,
       int(*compar)(const void *, const void *));
```

Huh?

qsort Parameters

```
void qsort(
   void *base,
   size_t nmemb,
   size_t size,
  int(*compar)(const void *, const void *)
);
```

- void * ~ Generic pointer type. Variables of all pointer types can be assigned to void* pointers & vice versa
- size_t ~ size type (usually unsigned int)
- base ~ start address of array to be sorted
- nmemb ~ number of elements
- size ~ size of an element (in bytes)
- compar ~ function that compares two elements

Sorting a String Array

```
#include <cstdlib>
#include <iostream>
// a points to a char pointer, so does b
// returns 0 if strings *a and *b are equal
// return <0 if string *a < string *b, >0 otherwise
int my_strcmp(const void *a, const void *b) {
  return strcmp( *(char**)a, *(char **)b);
}

void sort_strings(char *A[], int n) {
  qsort(A, n, sizeof(A[0]), my_strcmp);
}

int main() {
  char *A[] = { "b", "c", "ccc", "a" }; // array of pointers
  const int N = sizeof(A)/sizeof(A[0]);
  sort_strings(A, N);
  for (int i=0; i < N; ++i) std::cout << A[i] << " ";
}
   output: a b c ccc
```

Unions

```
union Shared {
   int   i;
   float f;
   char  c4[4];
};

Shared s;         // sizeof(s) = 4 !

s.f     = 3.5;    // store float value in s
s.c4[3]++;        // UGH! multiplies s.f by 4 on little-
                  // endian machines!!!
s.i     = 17;     // store int value in s

for (int i=0; i<4; ++i)
  cout << cout.put(c4[i]); // write s to cout (binary format)
```

- Space-saving struct (identical syntax)
- All data members are stored at the **same location (only works for simple types, dangerous!)**

Assignment Operator

```
class Foo {
   public:
   int x;
   Foo() { x = 0; }
   Foo &operator=(const Foo &y) {
      x = y.x;
      return *this; // returns a reference to the object
   }                // itself. this points to the object and
};                  // is implicitely known in member funcs.

Foo a, b;  // calls constructor
a = b;     // assignment operator called
Foo c = a; // copy constructor called in declaration!
```

- The assignment operator can be overloaded for classes
- Prototype for class X: X &operator=(const X &x);
- Default assignment: bit-copy (perhaps not what you want if class has pointer members!)
- Operator = should return reference to variable – this makes a = b = 0 is possible!


```
#include <iostream>
using namespace std;

class X {
public:
  X() { cout << "CONSTR" << endl; }
  X(const X &x) { cout << "COPY" << endl; }
  X &operator=(const X &x) { cout << "ASSIGN" << endl; return *this; }
  ~X() { cout << "DESTR" << endl; }
};

void g(X x) { cout << "g" << endl; }

int main() {
  X u;                                  output: CONSTR
  cout << "AAA" << endl;                        AAA
                                                COPY
  X v(u);                                       BBB
  cout << "BBB" << endl;                        COPY
                                                CCC
  X w = v;                                      ASSIGN
  cout << "CCC" << endl;                        DDD
                                                COPY
  v = u;                                        g
  cout << "DDD" << endl;                        DESTR
                                                EEE
  g(v);                                         DESTR x3
  cout << "EEE" << endl; }
```