07 - Lecture - data structure

Reading: K&R2, chapters 5 and 6

More on char**

Lab 2, part 2 revisited:

```
    int main(int argc, char **argv)
    {
        if (argc <= 1)
            return 1;

        char **copy = duplicateArgs(argc, argv);
        char **p = copy;

        argv++;
        p++;
        while (*argv) {
            printf("%s %s\n", *argv++, *p++);
        }

        freeDuplicatedArgs(copy);

        return 0;
    }
```

String functions in the C standard library

You need: #include <string.h>

See K&R2, p249-250 for a list, and see the man pages ("man 3 strlen", for example) for detailed descriptions.

Some examples:

- strlen(const char *)
- strcmp(const char *, const char *)

- strcpy(char *dest, const char *src)
- strncpy(char *dest, const char *src, size_t n)

    - Use strcpy() only when you KNOW dest points to memory >= strlen(src)+1. Otherwise, use strncpy() as follows:

      ```
      char buf[100];
      strncpy(buf, input, sizeof(buf) - 1);
      buf[sizeof(buf) - 1] = ’\0’;
      ```

- strcat(char *, const char *)
- strncat(char *, const char *, size_t)

- memcpy(void *, const void *, size_t)

- memset(void *, unsigned char, size_t)

BTW, what’s all that const all about?

```
const int BUF_SIZE = 1024;  // good alternative to #define

const int *p = &x;  // *p = 0 is an error, but p = &y is ok

int *const p = &x;  // *p = 0 is ok, but p = &y is an error

const int *const p = &x;  // neither *p = 0 nor p = &y is allowed
```

Function pointers

Motivation:

```
void qsort(void *baseAddress, size_t numElem, size_t sizeElem,
                int (*compareFn)(const void *, const void *));

int compareFloat(const void *v1, const void *v2)
{
    float x = *(float *)v1;
    float y = *(float *)v2;
    if (x < y)
        return -1;
    else if (x > y)
        return 1;
    else
        return 0;
}

int compareString(const void *v1, const void *v2)
{
    // What do we do here?  Is the following correct?
    //
    //     return strcmp((char *)v1, (char *)v2);
    //
    // If not, how do we fix it?
}

int main(int argc, char **argv)
{
    ...

    qsort(array_of_100_floats, 100, sizeof(float), &compareFloat);

    ...

    qsort(argv, argc, sizeof(char *), &compareString);

    ...
}
```

Declaration and usage:

```
    int (*f1)(const void *v1, const void *v2);
    int  *f2 (const void *v1, const void *v2);
    int (*f3[5])(const void *v1, const void *v2);

    float a = 1.0;
    float b = 2.0;

    f1 = &compareFloat;  // or simply f1 = compareFloat;
    int compareResult = (*f1)(&a, &b);  // or simply f1(&a,&b)

    // similarly, using array of function pointer f3:

    f3[0] = &compareFloat;
    int compareResult = (*f3[0])(&a, &b);
```

Complicated declarations

```
    char **argv
        argv: pointer to pointer to char
    int (*daytab)[13]
        daytab:  pointer to array[13] of int
    int *daytab[13]
        daytab:  array[13] of pointer to int
    void *comp()
        comp: function returning pointer to void
    void (*comp)()
        comp: pointer to function returning void
    char (*(*x())[])()
        x: function returning pointer to array[] of
        pointer to function returning char
    char (*(*x[3])())[5]
        x: array[3] of pointer to function returning
        pointer to array[5] of char
```

Struct

Similar to Java classes, except there are no methods in structs

Example:

```
        struct point {
            int x;
            int y;
        };

        struct point pt;

        pt.x = 2;
        pt.y = 3;
```

Or you can give it a synonym using typedef:

```
        typedef struct {
```

```
    int x;
    int y;
} Point;

Point pt;

pt.x = 2;
pt.y = 3;
```

Accessing struct members using pointer to struct:

```
struct point *pPt = &pt;

(*pPt).x = 2;

pPt->y = 3;
```

Structures are passed to (and returned from) a function by value (like everything else in C):

```
struct point getMidpoint(struct point p1, struct point p2);
```

When struct is large, it’s better to pass pointers (although in this particular case, it doesn’t really matter since struct point is small):

```
struct point getMidpoint(struct point *p1, struct point *p2);
```

Union
------

similar to struct, but all fields occupy the same memory location

example:

```
union value {
    unsigned int asInt;
    float        asFloat;
};

union value v;

v.asFloat = 3.14f;

// you can now examine the bit pattern using v.asInt
```

Self-referential structure

Commonly used to implement data structures such as linked lists:

```
struct IntNode {
    int data;
    struct IntNode *next;
};
```

```
        struct IntNode *head = NULL;
```

Or for holding floating point numbers:

```
        struct DoubleNode {
            double data;
            struct DoubleNode *next;
        };

        struct DoubleNode *head = NULL;
```

How do we write a generic linked list that works with any type?

In C, we use void*. (Lab assignment #3 is on this topic. We’ll do this in a better way in lab #10, after we learn C++.)

```
        struct Node {
            void *data;
            struct Node *next;
        };

        struct Node *head = NULL;
```