CS3157: Advanced Programming

Lecture #6

June 12

Shlomo Hershkop
shlomo@cs.columbia.edu

Outline

- Wrap up pointers
- File manipulations
- Working with text
- Wrapping up C
- Lab 3

- time permitting: shell programming

Announcements

- Will be posting homework project 2 this week
  - will be doing lab 3 today (due next class), no too long
  - lab 4 out Wednesday
  - Hw2 posted thursday
- If you are having a problem/idea/anything please stop by office hours

From last time

- Struct and typedefs
- How?
- Why?

Back to Pointers

- Pointers is what makes c so powerful

Array

- int a[4];
- a[0] = 2; a[2] = 6;
- a[1] = 3; a[3] = 4;

- a[i] is really *(a+i)
- Reason: a internally &a[0]
- Why can’t we say a++ ??

Pointers

- int *ptr = a;
- ptr[i] equals *(a+i)
- char c* = “this is fun”;
  - how to print this out?
  - can you say c[1] = ‘d’;
- What is wrong with the following:

```
*ptr = (int*)malloc(sizeof(int)*10);
```

  - try: gcc –Wall –o test test.c

automate this

- in c we can end up juggling many files
  - header, code, configuration, etc
- would like to use a tool to make all this automated
- make to the rescue

Makefile

- make is an executable which processes directions from a Makefile
- unix/cygwin, when type in ‘make’ will look for a Makefile (upper case M)
- directions are divided into rules

simple

```
sample1:        test.c
        gcc –o hw1 test.c

clean:
   rm –f *.o
```

not so simple

```
#welcome to my make file
CC = gcc
CLIBS = -lm
FLAGS = -Wall

sample1:            test.c
        ${CC} ${CLIBS} ${FLAGS} –o hw1 test.c

clean:
   rm –f *.o
```

Practical example

- Say I am going to take everyone's age in the room (example 30 students)….will input one at a time, and want a sorted list all the time
- can you write an outline of how it would work?
- Can you convert this to c ?

array

- Create the n size array
- Get number
- Figure where it goes
- Move everyone over to make place

- lets see some code

pointer

- one of the big advantages of pointers is that it gives your code a much neater look

- How would you write the same exact code with pointers?

Array allocation

- There is a difference between how java allocates a double array and how c/cpp do it…
- anyone know?
- which is better ?

Double Pointers

- Just to make sure you all caught this:
- We can have pointers to anything in memory
- why not point to a pointer ?
- int **ipp;
- int i = 5, j = 6; k = 7;
- int *ip1 = &i, *ip2 = &j;
- ipp = &ip1;

```
#include <stdlib.h>

  int allocstr(int len, char **retptr)
  {
     char *p = malloc(len + 1);
/* +1 for \0 */
     if(p == NULL)
          return 0;
     *retptr = p;
     return 1;
  }
```

```
char *string = "Hello, world!";
  char *copystr;
  if(allocstr(strlen(string), &copystr))
     strcpy(copystr, string);
  else    fprintf(stderr, "out of
  memory\n");
```

List ADT

- an ADT is an abstract data type
- we don’t worry about what goes underneath
- just interested in operations and idea
- we want to keep an ordered set of items and support:
  - insert
  - delete
  - find

Next

- lets also assume we want the list sorted at all times
- a simple implementation of a list is an array
- more complicated is a linked list
- how would you code a linked list in c ?

- First we define:

```
struct ELEMENT {int value; struct ELEMENT
  *next; };

 struct ELEMENT list;
list.next = (struct
  ELEMENT*)malloc(sizeof(struct ELEMENT));
```

- `list.value = 20;`

```
(*list.next).value = 22;
printf("val is %d\n",list.next->value);
```

Dealing with lists

- Usually easier to use a head object to start the list
- Options:
  - last node will be null (end of list)
  - Last node can link to first (easier to traverse)
  - Add back links to allow faster walkthroughs

compare

- So if I have 10 items
- What is the difference between an array and linked list?

Working with lists

```
void add(llnode **head, int data_in) {
llnode *tmp;
if ((tmp = malloc(sizeof(*tmp))) == NULL){
ERR_MSG(malloc);
void)exit(EXIT_FAILURE);
}

tmp->value = data_in;
tmp->next = *head;
*head = tmp;
}
```

```
/* ... inside some function ... */
llnode *head = NULL;
......
add(&head, some_data);
```

Reminder

- Your mother is right when she told you clean after yourself!
- How to clean up the list?

```
void freelist(llnode *head) {
llnode *tmp;

while (head != NULL) {
free(head->data);
tmp = head->next;
free(head); head = tmp;
}

}
```

Text based programming

- Many application of computer science

- Spell checking
- Learning
- Modeling
- compression

- important study of running time on algorithms before even bothering to code them

sample application

- one of the powers of being able to manipulate raw memory is the ability of getting your code super optimized
- i.e. instead of taking 24 hours to solve a problem can get it down to something faster:
  - 4 minutes ☺

Compression

- Anyone know how compression works ?

- On the computer text (characters) are represented fix length set of bits
- 7 bits for ASCII
- Can we do better than that?

Compression

- If we can use less bits for higher occurring characters, overall we will use less bits in our text file

Binary tree

- Let me introduce a data structure to you
- A binary tree has a node with optional left and right children
- Think of it as a linked list with two links

Hoffman compression

1. Create a frequency count of each of your characters in your file
2. Start to build a binary tree always combining 2 lowest frequencies into one tree the resulting frequency is the combined frequencies
3. Going left is 0, going right is 1

Example

- If I counted:
- E = 29
- A = 14
- T = 10
- B = 4
- D = 2
- C = 1

decompression

- So seeing a code, we simply run down the tree
- As soon as we hit a leaf, translate to that character

Compressing text

- How would you use Huffman to compress text??

Stream

- a stream is just a simple way of looking at any device
- stream can be file, screen, port, printer, keyboard etc
- open: hooks a pointer with the stream
- close: unhooks

Modes

- r Open a text file for reading
- w Create a text file for writing
- a Append to a text file
- rb Open a binary file for reading
- wb Open a binary file for writing
- ab Append to a binary file
- r+ Open a text file for read/write
- w+ Create a text file for read/write
- a+ Append or create a text file for read/write
- r+b Open a binary file for read/write
- w+b Create a binary file for read/write
- a+b Append a binary file for read/write

```
FILE *fp;

if ((fp = fopen("myfile", "r"))
  ==NULL){
   printf("Error opening file\n");
   exit(1);
}
```

Reminder

- make sure you are getting a valid pointer back
- make sure you are creating the correct mode

- why this is important

- int fclose(FILE *fp);

- The fclose() function closes the file associated with fp, which must be a valid file pointer previously obtained using fopen()
- disassociates the stream from the file
- The fclose() function returns 0 if successful and EOF (end of file) if an error occurs.

binary vs text

- Generally two modes OS operates in
- text
  - ascii
- binary
  - byte level

```
#include <stdio.h>  /* header file  */
#include <stdlib.h>
void main(void)
{

 FILE *fp;   /* file pointer */
 int i;

 /* open file for output */
 if ((fp = fopen("myfile", "w"))==NULL){
  printf("Cannot open file \n");
  exit(1);
 }
 i=100;

 if (fwrite(&i, 2, 1, fp) !=1){
  printf("Write error occurred");
  exit(1);
 }
 fclose(fp);

 /* open file for input */
 if ((fp =fopen("myfile", "r"))==NULL){
  printf("Read error occurred");
  exit(1);
 }
 printf("i is %d",i);
 fclose(fp);
}
```

other stuff

- int remove(char *file-name);
- void rewind(FILE *fp);

File manipulations

- `FILE *fopen (const char *path, const char *mode);`
- `FILE *Fp;`
- `Fp = fopen("/home/johndoe/input.dat", "r");`
- `fscanf(Fp, "%d", &x);`
- `fprintf(Fp, "%s\n", "File Streams are cool!");`
- `int fclose( FILE *stream );`

Command line arguments

- Many times you want to pass in specific information to your program as command line args

- Tool for helping you do this:

```
int getopt(int argc, char * const argv[], const char
  *optstring);

extern char *optarg;


extern int optind, opterr, optopt;
```

Change main method

- `int main(int argc, char **argv)`

- `./junk -b something data.txt`

```
int ich;

  while ((ich = getopt (argc, argv, "ab:c")) != EOF) {
    switch (ich) {
      case 'a': /* Flags/Code when -a is specified */
        break;
      case 'b': /* Flags/Code when -b is specified */
                /* The argument passed in with b is specified */
               /* by optarg */
        break;
      case 'c': /* Flags/Code when -c is specified */
        break;
      default: /* Code when there are no parameters */
        break;
    }
  }

  if (optind < argc) {
    printf ("non-option ARGV-elements: ");
    while (optind < argc)
      printf ("%s ", argv[optind++]);
    printf ("\n");
  }
```

wrapping up c

- c is very powerful language
- because of advanced in hardware/software push today to write OO code
- for many reasons:
  - re-usability
  - modularity
  - scalability
  - maintainability
- Need to know c
  - many good ideas first implemented here
  - might need to maintain code in c
  - might end up writing a specific

Lab 3

- we will be doing lab 3 in class today….., but you need to submit your own COMMENTED work
- notice the comments

For Wednesday

- Read up on c file handling
- Read up on structs, linked lists, nodes, huffman algorithm
- Wrap up the lab

- Get c++ book and read intro parts on language and basic usage