CS3157: Advanced Programming

Lecture #5
June 11
Shlomo Hershkop
shlomo@cs.columbia.edu
Overview

- Today:
  - Wrapping up c
    - Some basics..
    - Long example with everyone’s favorite part of c! pointers, arrays and puzzles
Project 1

- We did a chunk in last class

- Make sure to start early
Reminder from last time

- When creating variables, can use built-in primitive types such as int, double, float
- Or can create new types by using struct keyword to wrap basic types into new types
example

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

- **Usage:**
  ```c
  struct point a;
  a.x = 5;
  a.y = 10;
  ```

- So in memory have a single location of sizeof(struct point)
Putting it to good use

- Lets practice by working with a puzzle game

- MAGIC SQUARE!

- Anyone ever heard of this?
Idea

- A magic square is a N by N matrix filled with integers in the range \([1 – N^2]\) so that each entry in the matrix has a unique value.
- The sum of each of the rows and each of the columns and each of the two diagonals must be equal to some number (when \(n=3\) it is 15).
- Note that there is not a single solution to the magic square problem.
Lets get started

- Maintest1.c
- Magicsquare1.c
- Magicsquare1.h
Magicsquare.h

#define MSSize 3

typedef int MSQUARE_TYPE[MSSize][MSSize];

typedef MSQUARE_TYPE * MagSquare_PTR;

void initSquare( MagSquare_PTR ,int);

void printSquare( MagSquare_PTR,int );
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "magicsquare1.h"

int main() {
    MagSquare_PTR mptr = /* replace with something with malloc */;

    srand( time( NULL ));

    initSquare( mptr, MSSIZE );

    printSquare( mptr, MSSIZE);

} // end of main()
Magicsquare1.c

Define 2 functions

initSquare(MagSquare_PTR, int size)

printSquare(MagSquare_PTR, int size)
code

- Lets do the initsquare together on the board line by line

- Very simple function to put stuff in...
  - Would like random assignment to the array
  - Can do it by creating a $1-n^2$ array, when use one of the numbers replace by -1...if we hit -1 read sequentially until hit non -1 (wrapping around if necessary)
  - Else just assign 1-$n^2$

- Make sure to ask as we go...
next

- How would the print square work ??

- Lets do it together
Compile and test

Should be able to create a $n \times n$ square of random number ordered and print it out...
Add specific functions

- We will code some functions to help compute a magic square....don’t forget to code function declarations in the .h file
int sumColumn( int column, MagSquare_PTR, int MSSize )

The function takes the 2-dimensional magic square array and a column number and returns the sum of the N values in that column.

Lets code it and compile it now....
int sumRow(int row, MagSquare_PTR, int size)

- As before sums the row
- Code and compile
The function takes the 2-dimensional magic square array and a diagonal number and returns the sum of the n values in that diagonal.

Use diagonal = 0 to refer to the diagonal that runs from the northwest corner down to the southeast corner. Use diagonal = 1 to refer to the diagonal that runs from the northeast corner down to the southwest corner.

Code and compile
Next

- Now we are ready to write test function to see if we have a magic square.

- Create a function called `int isMagic(MagSquare_PTR, int MSSIZE, int TOTAL)` which takes the magic square pointer as an argument and checks each of the rows and columns and diagonals to see if all the sums equal 15 (when n=3).

- The function should return 0 if the argument is not a magic square and 1 if it is a magic square.
Testing...

- Modify the above main() so that it calls isMagic() and prints out whether the square is a magic square or not.
Sample makefile

CC = gcc
CCFLAGS = -Wall -lm

step1_run: maintest1.c magicsquare1.o
   $(CC) $(CCFLAGS) -o test1 maintest1.c  magicsquare1.o
   ./test1

magicsquare1.o:  magicsquare1.c  magicsquare1.h
   $(CC) $(CCFLAGS) -c  magicsquare1.c

clean:
   rm -f *.o test1
Assuming you have it all working

How would you get it to make a non magic square into a magic square ???
Create a function called `permuteSquare(MagSquare_PTR, int size)` which takes the magic square pointer as an argument and randomly switches two entries in the array.

Do this by randomly picking two sets of row and column indices (in the range `[0 – (n-1)]`) and then swapping the entries located at each pair of indices. Check to make sure you aren’t picking the same spot .. that is swapping the same location with itself.
Wrapping up…

- Modify the main() so that after it calls isMagic(), if the square is not magic, then it calls permuteSquare() to switch around two entries in the square and then test again to see if the square is magic.
- Do this repeatedly until a magic square is found. When a magic square is found, call printSquare() again to print out the magic square.
- Have the program count the number of times it has to permute the square in order to find a magic square.
- Print that out too (Hint: count it where you call method permutesquare).
submission

- Magic square is your lab 3....so please create a readme and submit the files once you include all comments etc....
  - By June 17th..
Other stuff…

- Now that you have mastered pointers and basic C
- Let's talk about the running program itself
A process is basically a single running program. It may be a "system" program (e.g. login, update, csh) or program initiated by the user (magicsquare).

When UNIX runs a process it gives each process a unique number - a process ID, pid.

The UNIX command ps will list all current processes running on your machine and will list the pid. (try ps –aux in cygwin or unix)

The C function int getpid() will return the pid of process that called this function.

A program usually runs as a single process.
system

- Can execute command side program by using system command

```c
main()
{
    printf("Files in Directory are:\n");
    system("ls -l");
}
```
Shell related stuff

- We will be coming back to shell based programming

- Will just review some basics
redirection

- stdin, stdout and stderr may be redirected during shell execution of your code
- < redirects stdin (0) to come from a file
- > redirects stdout (1) to go to file

Example:
- ./magicsquare.exe > output.txt will take the output of your program and put it into the output.txt file
- ./sometest < inputsample.txt will pass the text file as stdin...
Redirection II

- `>>` appends stdout to the end of a file
  - So don’t overwrite the output file
- `&>` redirects stderr (2)
- `>>&1` sends stderr to the same place that stdout is going
More programs you might like

- Cal (can get it installed with cygwin)
  - Prints a calendar

```
bash-2.05$ cal 2 2004
February 2004
Su Mo Tu We Th Fr Sa
 1 2 3 4 5 6 7
 8 9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29
```
Usage stuff

■ df
bash-2.05$ df -h
<table>
<thead>
<tr>
<th>Filesystem</th>
<th>Size</th>
<th>Used</th>
<th>Avail</th>
<th>Use%</th>
<th>Mounted on</th>
</tr>
</thead>
<tbody>
<tr>
<td>/dev/hda3</td>
<td>197M</td>
<td>157M</td>
<td>31M</td>
<td>84%</td>
<td>/</td>
</tr>
<tr>
<td>/dev/hda7</td>
<td>296M</td>
<td>65k</td>
<td>280M</td>
<td>1%</td>
<td>/tmp</td>
</tr>
<tr>
<td>/dev/hda5</td>
<td>2.4G</td>
<td>2.0G</td>
<td>385M</td>
<td>84%</td>
<td>/usr</td>
</tr>
</tbody>
</table>

■ du
bash-2.05$ du -ch code2
48k     code2/ai1
56k     code2
56k     total

■ quota
Useful programs

Here is a list of useful utilities standard in many unix like shells
Useful tools & commands

- wc
- grep
- cut
- cat
- sort
- sed
- ps
- who
- finger
- kill
- touch
- Test
- diff
Global Regular Expression Parser
GREP

- one of the most useful tools in unix – make sure its your friend

- three standard versions:
  - plain old grep
  - extended grep: egrep
  - fast grep: fgrep

- used to search through files for ... regular expressions!

- prints only lines that match the given pattern

- a kind of filter

- BUT it’s line oriented
input can be one or more files or can be piped into grep

examples:
grep "^[aeiou]" myfile
ls -l | grep t

useful options:
- ignore case
- match pattern as a word
- return only the filename if there’s a match
- reverse the normal action (i.e., return what doesn’t match)
- examples:
  grep -i "^[aeiou]" myfile
  grep -v "^[aeiou]" myfile
  grep -iv "^[aeiou]" myfile

- how do you list all lines containing a digit?
- how do you list all lines containing a 5?
- how do you list all lines containing a 0?
- how do you list all lines containing 50?
- how do you list all lines containing a 5 and an 0?
sort

- sorts lines of input

- input can be a file or a piped command (see below)

- three modes: sort, check (sort -c), merge (sort -m)

- syntax: sort <-t> <-n> <-r> <-o> POS1 -POS2+
  - -n Numeric
  - -M sort months Jan < Dec
  - -f ignore case
  - -r reverse the sort

- note that POS starts with 0; default delimiter is whitespace
Question?

- So what does:

  - `ls -la | sort -n`
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");
}
Now …c++

very brief history...

- C was developed 69-73 at Bell labs.
- C++ designed by Bjarne Stroustrup at AT&T Bell Labs in the early 1980’s
- originally developed as “C with classes”
- Idea was to create reusable code
- development period: 1985-1991
- ANSI standard C++ released in 1991
Four main OOP concepts

- **abstraction**
  - creation of well-defined interface for an object, separate from its implementation
  - e.g., Vector in Java
  - e.g., key functionalities (init, add, delete, count, print) which can be called independently of knowing how an object is implemented

- **encapsulation**
  - keeping implementation details “private”, i.e., inside the implementation

- **hierarchy**
  - an object is defined in terms of other objects
  - Composition => larger objects out of smaller ones
  - Inheritance => properties of smaller objects are “inherited” by larger objects

- **polymorphism**
  - use code “transparently” for all types of same class of object
  - i.e., “morph” one object into another object within same hierarchy
Main difference between c and cpp

- C’s power is driven by functions. You define a set of function which operate in a specific sequence to implement some algorithm
  - Top down
- CPP is an object oriented language
  - Bottom up
Compatible

- Cpp is backwards compatible with c
- Cpp is bottom up approach
- Cpp compilers will compile c code
Advantages

- There are a bunch of (claimed) advantages to using CPP over C.
Advantages

- Can create new programs faster because we can reuse code
- Easier to create new data types
- Easier memory management
- Programs should be less bug-prone, as it uses a stricter syntax and type checking.
- `Data hiding', the usage of data by one program part while other program parts cannot access the data
- Will whiten your teeth
So because C++ is built on the idea of objects, the code is a lot easier to read.

Should be much easier to learn since you are familiar with Java.

Here is a sample program:
Hello.cpp

#include <iostream.h>
#include <stdio.h>
using namespace std;

main() {
    cout << "hello world\n";
    cout << "hello" << " world" << endl;
}

compile using:
g++ hello.cpp -o hello

like gcc (default output file is a.out)
For Wednesday

- Look over notes so far
  - Please write down any questions you might have
  - Start project 1

- Start reading c++ section