CS3157: Advanced Programming

Lecture #12

Apr 2

Shlomo Hershkop
shlomo@cs.columbia.edu

Announcements

- Emailed about projector….hope it actually works
- Reminder: off this Wednesday and Monday next week to work on homework
- Back next Wednesday (apr-11) for lab

Today

- Random : text compression
- Wrap up c
  - getopt

- How to learn a new language
- Starting with C++

Compression

- Anyone know how compression works ?

Basic idea

- Find a pattern
- Replace long pattern with shorter one
- Get smaller file which is packed

- To unpack
- Replace short pattern with longer ones

- Main question: how to find and represent patterns in the data

Security issue

- One of the main question we can ask:
- WHAT CAN GO WRONG ??

Example

- 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

- Huffman compression

- Higher frequencies, let us use less bits

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??

By the way…

- `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 to process text, 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

- We have dealt with very basic command line args
- 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 */
         someptr = 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 function to run quickly/efficiently
  - Good to debugging technology , since many things built on top of c/c++ principles

Moving on

- Main idea of the course
  - Teach you skills needed in programming environment
  - We are covering some useful languages
  - But
  - Should understand how to learn a new language
    - Quickly
    - Correctly

No magic bullet

- Learn basics of syntax
- Learn basics of compiler/translator
- Learn how to debug ← ← ← most people miss this!
- Find comfortable environment ← people miss this too!

- Practice
- Practice
- Practice
- See how others are using the language

Language learning

- Languages based on known (to you) paradigm can be learned relatively quickly
- Assuming you know c and java
- Should be able to program c++ within hour

Language learning

- Languages based on unknown (to you) paradigm can harder to learn
- Anyone familiar with prolog ??

Sample prolog program

```
split(H, [A|X], [A|Y], Z) :-
  order(A, H), split(H, X, Y, Z).
split(H, [A|X], Y, [A|Z]) :-
  not(order(A, H)), split(H, X, Y, Z).
split(_, [], [], []).
quicksort([], X, X).
quicksort([H|T], S, X) :-
  split(H, T, A, B),
  quicksort(A, S, [H|Y]),
  quicksort(B, Y, X).
```

Bottom line

- Many programming languages out there
- Don’t spend a lifetime learning them all ☺
- Not all the same
- Not all different

- Try to understand why a language was created

Outline for c++

- Background OOP
- c++ stuff:
  - Language basics: identifiers, data types, operators, type conversions, branching and looping, program structure
- data structures: arrays, structures
- pointers and references differences
- I/O: writing to the screen, reading from the keyboard, iostream library
- classes: defining, scope, ctors and dtors

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
  - design parts of the system
  - put them together
  - bottom up approach

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