Introduction to Computer Science and Programming in C

Session 11: October 7, 2008

Columbia University

Announcements

- Homework 2 is out. Due 10/14 before class
- Midterm Review on 10/16, exam on 10/21
- Bert's office hours on 10/14 moved to Wednesday 10/15, 1-3 PM (or by appointment)

Review

- File I/O – use the FILE type
  - fprintf(), printf(), sprintf() – print formatted text to file, stdout, string
  - fscanf(), sscanf() – read formatted text from file, string into variables
  - fgets(), fgetc(), fputs(), fputc() – get string/char from file, put string/char in file.

Today

- Leftovers from FILE I/O:
  ferror(), feof(), rewind()
- C preprocessor

ferror()

- Since reading files communicates with OS, there can be more errors outside your control.
- Check for errors in your FILE variables with the function

```
int err = ferror(myFile);
```

- Returns non-zero if error occurs.
  Zero if everything is fine.

feof()

- Similarly, we can check if a FILE variable has reached the end of a file without calling fscanf() or fgets().
- Instead, use
  `while(feof(myFile)==0)`
- feof() returns non-zero if EOF has occurred.

rewind()

- When we want to start reading a file at the beginning, use rewind()

```
rewind(myFile);
fscanf(myFile, “%f\n”, &x);
```

- Similar to calling fclose() and reopening file, but instead the OS won’t let someone else grab the file.

C preprocessor

- Special text that gets preprocessed by compiler
- Essentially modifies your code right before compiling.
- Preprocessor commands always begin with #

#include

- `#include <stdio.h>`
  Copies the text in stdio.h into your code, including function definitions for printf(), etc.
- `#include "myFile.h"`
  copies local file myFile.h into current program
- Use <> brackets for standard library (built in C files), "" quotation marks for local files (your own files)

#define

- `#define VALUE 10`
  Macro replaces all occurrences of the string VALUE in your program with 10.

- Useful for defining constants:
```
#define MAX_NAME 30
char name[MAX_NAME];
```

- Preprocessor does not check syntax for these replacements: we can do very weird stuff

#define

- ```
  #define FOR_ALL for (i=0; i<ARRAY_SIZE; i++)
  FOR_ALL {
      data[i] = 0;
  }
  ```

- #**define** just replaces text.

- ```
  #define FIRST     7
  #define SECOND    5
  #define BOTH      FIRST+SECOND
  int main() {
      printf("The square of both parts is %d\n",
          BOTH * BOTH);
      return 0;
  }
  ```

#define

- Especially useful for setting constant array sizes.
- Some compilers not allow you to define an array with a const variable size:

```
int size;
int A[size]; /* should cause error */
```

- ```
  const int size=10;
  int A[size]; /* causes errors in many compilers */
  ```
- ```
  #define SIZE 10;
  int A[SIZE]; /* OK in any C compiler */
  ```

Conditional Compilation

- `#ifdef WORD`
- Checks if WORD is defined as a macro in the preprocessor. If so, keep the following lines until `#endif`
- Also `#ifndef`, `#else` and `#undef`

#ifdef Examples

- ```
  #define DEBUG
  ...
  #ifdef DEBUG
      printf("The value of x is %d\n", x);
  #endif
  ```

- ```
  #ifndef _HELPER_INCLUDED
  #include "helper.h"
  #define _HELPER_INCLUDED
  #endif
  ```

#ifdef Examples

- ```
  /*** COMMENT OUT THIS SECTION
  startFunction();
  counter++; /* increase the count */
  finishFunction();
  *** END OF COMMENTED SECTION **/
  ```

- ```
  #undef DEBUG
  #ifdef DEBUG
  startFunction();
  counter++; /* increase the count */
  finishFunction();
  #endif
  ```

Parameterized Macros

- #define can also create macros with arguments
- `#define SQR(x) (x * x)`
- Like a function, but just replaces text.

Viewing the preprocessed code

- GCC has a special flag to just run the preprocessor
- gcc -E myFile.c
- If output is too long, we can send output to a file using >
  gcc -E myFile.c > output.txt
- Saves gcc's output to output.txt