Introduction to Programming Systems

CS 217

Goals

- Master the art of programming
  - learn how to be “good” programmers
- Learn C and the Unix development tools
  - C is the systems language of choice
  - Unix has a rich development environment
- Introduction to computer systems
  - machine architecture
  - operating systems
  - compilers

Outline

- First four weeks
  - C programming
- Second four weeks
  - Machine architecture
- Third four weeks
  - Unix operating system

Coursework

- Six programming assignments (60%)
  - string library
  - collection ADT
  - table ADT
  - Sparc assembly
  - assembler
  - shell
- Two midterms (30%)
  - 5th week
  - 10th week
- Class participation (10%)

Materials

- "Required" textbooks
  - C: A Reference Manual. Harbison & Steele
  - SPARC Architecture, etc. Paul
- "Recommended" textbooks
  - C Interfaces and Implementations. Hanson
  - Programming with GNU Software. Loukides & Oram
- Other textbooks
  - The C Programming Language, Kernighan & Ritchie
  - C Programming: A Modern Approach, King
- Web pages
  - `www.cs.princeton.edu/courses/cs217/`

Facilities

- Unix machines
  - CIT's **`arizona`** cluster
  - SPARC lab in Friend 016
- Your own laptop
  - **`ssh`** access to **`arizona`**
  - run GNU tools on Windows
  - run GNU tools on Linux

Logistics

- Lectures
  - MW 10AM, CS102, 10:00AM
  - introduce concepts
  - work through programming examples
- Precepts
  - Tuesdays & Thursdays, 10:00AM or 1:30PM
  - demonstrate tools (gdb, makefiles, emacs, …)
  - work through programming examples

Can we have only one precept?

Software is Hard

"What were the lessons I learned from so many years of intensive work on the practical problem of setting type by computer? One of the most important lessons, perhaps, is the fact that SOFTWARE IS HARD. From now on I shall have significantly greater respect for every successful software tool that I encounter. During the past decade I was surprised to learn that the writing of programs for TeX and Metafont proved to be much more difficult than all the other things I had done (like proving theorems or writing books). The creation of good software demands a significantly higher standard of accuracy than those other things do, and it requires a longer attention span than other intellectual tasks."

Donald Knuth, 1989

Software in COS126

Specification → Design → Programming → Debugging → Testing

1 Person
10^2 Lines of Code
1 Type of Machine
0 Modifications
1 Week

Software in the Real World

Specification

Design

Programming

Debugging

Testing

Lots of People
10^6 Lines of Code
Lots of Machines
Lots of Modifications
1 Decade or more

Good Software in the Real World

- Understandable
  - Well-designed
  - Consistent
  - Documented

  ← Write code in modules with well-defined interfaces
- Robust
  - Works for any input
  - Tested

  ← Write code in modules and test them separately
- Reusable
  - Components

  ← Write code in modules that can be used elsewhere
- Efficient
  - Only matters for 1%

  ← Write code in modules and optimize the slow ones

Modules

- Programs are made up of many <u>modules</u>
- Each module is small and does one thing
  - string manipulation
  - mathematical functions
  - set, stack, queue, list, etc.
- Deciding how to break up a program into modules is a key to good software design

Interfaces

- An interface defines what the module does
  - decouple clients from implementation
  - hide implementation details
- An interface specifies…
  - data types and variables
  - functions that may be invoked

```
StringList *StringList_new(void);
void StringList_free(StringList *list);

void StringList_insert(StringList *list, char *string);
void StringList_remove(StringList *list, char *string);
void StringList_print(StringList *list);

int StringList_getLength(StringList *list);
```

Implementations

- An implementation defines how the module does it
- Can have many implementations for one interface
  - different algorithms for different situations
  - machine dependencies, efficiency, etc.

```
StringList *StringListCreate(void)
{
  StringList *list = malloc(sizeof(StringList));
  list->entries = NULL;
  list->size = 0;
}

void StringListDelete(StringList *list)
{
  free(list);
}

etc.
```

Clients

- A client uses a module via its interface
- Clients see only the interface
  - can use module without knowing its implementation
- Client is unaffected if implementation changes
  - as long as interface stays the same

```
int main()
{
  StringList *list = StringListCreate();
  StringListInsert(list, "CS217");
  StringListInsert(list, "is");
  StringListInsert(list, "fun");
  StringListPrint(list);
  StringListDelete(list);
}
```

Clients, Interfaces, Implementations

- Interfaces are contracts between clients and implementations
  - Clients must use interface correctly
  - Implementations must do what they advertise

Interface

Client

Implementation

- Examples from real world?

Clients, Interfaces, Implementations

- Advantages of modules with clean interfaces
  - de-couples clients from implementations
  - localizes impact of change to single module
  - allows sharing of implementations (re-use)
  - allows separate compilation
  - improves readability
  - simplifies testing
  - etc.

```
int main()
{
  StringList *list = StringListCreate();
  StringListInsert(list, “CS217”);
  StringListInsert(list, “is”);
  StringListInsert(list, “fun”);
  StringListPrint(list);
  StringListDelete(list);
}
```

C Programming Conventions

- Interfaces are defined in header files (.h)

stringlist.h

```
StringList *StringListCreate(void);
void StringListDelete(StringList *list);
void StringListInsert(StringList *list, char *string);
void StringListRemove(StringList *list, char *string);
void StringListPrint(StringList *list);
int StringListGetLength(StringList *list);
```

C Programming Conventions

- Implementations are described in source files (.c)

stringlist.c

```
#include "stringlist.h"


StringList *StringListCreate(void)
{
  StringList *list = malloc(sizeof(StringList));
  list->entries = NULL;
  list->size = 0;
}

void StringListDelete(StringList *list)
{
  free(list);
}

etc.
```

C Programming Conventions

- Clients "include" header files

main.c

```
#include "stringlist.h"

int main()
{
  StringList *list = StringListCreate();
  StringListInsert(list, "CS217");
  StringListInsert(list, "is");
  StringListInsert(list, "fun");
  StringListPrint(list);
  StringListDelete(list);
}
```

Standard C Libraries

`assert.h`	assertions
`ctype.h`	character mappings
`errno.h`	error numbers
`math.h`	math functions
`limits.h`	metrics for ints
`signal.h`	signal handling
`stdarg.h`	variable length arg lists
`stddef.h`	standard definitions
`stdio.h`	standard I/O
`stdlib.h`	standard library functions
`string.h`	string functions
`time.h`	date/type functions

Standard C Libraries (cont)

- Utility functions **`stdlib.h`**
  ```
  atof, atoi, rand, qsort, getenv,
  calloc, malloc, free, abort, exit
  ```
- String handling **`string.h`**
  ```
  strcmp, strncmp, strcpy, strncpy, strcat,
  strncat, strchr, strlen, memcpy, memcmp
  ```
- Character classifications **`ctypes.h`**
  ```
  isdigit, isalpha, isspace, isupper, islower
  ```
- Mathematical functions **`math.h`**
  ```
  sin, cos, tan, ceil, floor, exp, log, sqrt
  ```

Example: Standard I/O Library

- **`stdio.h`** hides the implementation of "**`FILE`**"

```
extern FILE *stdin, *stdout, *stderr;
extern FILE *fopen(const char *, const char *);
extern int fclose(FILE *);
extern int printf(const char *, …);
extern int scanf(const char *, …);
extern int fgetc(FILE *);
extern char *fgets(char *, int, FILE *);
extern int getc(FILE *);
extern int getchar(void);
extern char *gets(char *);
. . .
extern int feof(FILE *);
```

Summary

- We will learn good programming in first third of this class
- A key to good programming is modularity
  - A program is broken up into meaningful modules
  - An interface defines <u>what</u> a module does
  - An implementation defines <u>how</u> the module does it
  - A client sees only the interfaces, not the implementations
- First assignment is to provide the implementation of the standard C string manipulation module defined in **`string.h`**