

Getting Started

Project 1

Shell?

- Interpreter for a simple programming language
 - Can interface with directly
 - Or run a file called a shell script
- Usually used to quickly interface with an operating system

Shell Examples

- sh – The first shell, came with Unix
- csh – The C-shell
- ksh – The Korn shell
- tchsh – The Tenex C-shell, used on linprog
- bash – The Bourne Again Shell (default on most Linux distributions)
- DOS/cmd – The Windows Shell

Shell Preparation

- These next few lectures will walk you through how to build a shell
- The first thing to cover is C

C Standard Library

- Provides a standard way to:
 - Open / close files
 - Read / write data
 - Manipulate and compare c-strings
 - Convert c-strings to other types (and vice-versa)
 - Allocate/free memory
 - Sort/search input
 -

Opening and Closing Files

- `#include <stdio.h>`
 - `FILE *fopen(const char *file_name, const char *mode)`
 - `void fclose(FILE *file)`
- FILE pointers provide access to the contents of a file

Opening and Closing Files

- Modes:

- r – Read. Error if file does not exist
- w – Write. Replaces existing file or creates a new file
- a – Append. Add data to existing file or creates a new file
- w+ – Equivalent to both 'r' and 'w'
- a+ – Equivalent to both 'r' and 'a'

C-strings

- C functions using strings, require the string to be null-terminated
- That is the final character in the string needs to be '\0'
 - Otherwise, the function will extend beyond the bounds of the string
- If there are any other '\0' characters within the string, then you can not use these functions
 - An example would be a 'string' containing raw data

Writing Output

- `#include <stdio.h>`
 - `int printf(const char *format, ...)`
 - `int sprintf(const char *buffer, const char *format, ...)`
 - `int fprintf(FILE *stream, const char *format, ...)`
 - `int fputs(char *str, FILE *stream)`
- Takes in a c-string and format specifiers to format the output

Writing Output

- `sprintf` writes to a buffer
- `fprintf` and `fputs` write to a file
- The return value is the number of characters written
 - Null character implicitly added in `sprintf` is not counted

Format Specifiers

- Most I/O functions in the C Standard Library use format specifiers and flags
- Common specifiers:
 - %d - signed integer value
 - %u - unsigned integer value
 - %f - float value
 - %x - hexadecimal value
 - %c - character value
 - %s - string value

Format Specifiers

- Typing *man printf* in a shell will give a more complete list

- Example

```
printf("%s %d\n", "Project due: February", 3);
```

```
Project due: February 3
```

Reading Input

- `#include <stdio.h>`
 - `int scanf(const char *format, ...)`
 - `int sscanf(const char *buffer, const *format, ...)`
 - `int fscanf(FILE *stream, const char *format, ...)`
 - `int fgets(char *buffer, int num, FILE *stream)`
- Scanf functions return number of **items** read
- Fgets reads *num* characters from a file into *buffer*
 - returns the number of **characters** read

C-string Comparison

- `#include <string.h>`

- `int strncmp (const char *str1, const char *str2, size_t num)`
 - `int strcmp (const char *str1, const char *str2)`

- Returns

- `<0` if *str2* contains the large value at the first non-matching character
 - `0` if value of *str1* == value of *str2*
 - `>0` if *str1* contains the large value at the first non-matching character

C-string Comparison

- Do **not** do *if(str1 == str2)*
 - This is a pointer comparison, not a value comparison

C-String Copying

- `#include <string.h>`
 - `char *strncpy(char *dest, const char *src, size_t num)`
 - `char *strcpy(char *dest, const char *src)`
- Copies source string into destination string
- Returns the pointer to dest string
- Make sure to allocate enough room for dest string
- Again, do not do *dest = src*

C-string Searching

- `#include <string.h>`
 - `char *strstr(const char *pattern, const char *string)`
 - `char *strchr(const char character, const char *string)`
- Search for the first occurrence of a pattern/character in a string
- Returns the starting address of the target item
 - Null if not found

Memory Allocation

- `#include <stdlib.h>`
 - `void *malloc(const size_t num_bytes)`
 - `void *calloc(const size_t num_objs, const size_t obj_size)`
 - `void free(void *obj)`
- Need to use when you don't know the size ahead of time

Memory Allocation

- Need to cast malloc, calloc to desired type
 - e.g. *char *str = (char *)malloc(sizeof(char) * num_chars);*
- calloc returns a 0-initialized pointer
 - Recommended over malloc
- free deallocates dynamically allocated memory

Potential Problems with Free

```
void memory_leak(int size) {  
    int *leak = (int *)calloc(size, sizeof(int));  
    return;  
}
```

/* Never freed! Can not access after leaving
function because no pointer returned! */

Potential Problems with Free

```
void dangling_reference(int size) {  
    int *reference = (int *)malloc(size * sizeof(int);  
    free(ref);  
  
    /* Already freed! Should not access! */  
    printf("%d\n", ref[0]);  
    return;  
}
```

Tools

Man Pages

- Documentation that comes with most Unix-like systems
- Contains information for C functions, packages, bash commands, system calls, etc
 - Examples
 - *man bash*
 - *man strncpy*
 - *man bsearch*

Man Pages

- When there are multiple definitions, it will refer to the lower section
 - Use *man 3 printf* to see C version
 - Otherwise it will show bash version (*man 1 printf*)
- Section information can be found at *man man*

tar

- Tape ARchiver
- To archive
 - *tar -cvf tarfile.tar files to tar*
- To extract
 - *tar -xvf tarfile.tar*
- For gzipping
 - Use 'z' flag and .gz extension
 - *tar -xvzf tarfile.tar.gz*

Make

- Automated software build system
- You'll use it provide a simple way to the executable for your project
 - Name it: “Makefile”
- It works by specifying a target, what it depends on, and how to transform the dependencies
- In general, it looks like:

target : dependency1, dependency2, ...

command1

command2

command3

Make Example

CC=gcc

CFLAGS=-I. -ansi -pedantic -Wall

.PHONY : compile clean run

compile : main.x

main.x : main.o util.o

<tab>\$(CC) \$(CFLAGS) -o hello.x hello.o