Course Recap

CS 2022: Introduction to C

Instructor: Hussam Abu-Libdeh

Cornell University
(based on slides by Saikat Guha)

Fall 2011, Lecture 13

Hello World!

```
#include <stdio.h>

void print_greeting()
{
  printf("Hello World!\n");
}

int main(int argc, char **argv)
{
  print_greeting();
  return 0;
}
```

Command Line Arguments

- When an application launches, the operating system can pass it *command line arguments*
- Optional and not required
- `int main(int argc, char **argv)`
  - `argc` - arguments count
  - `argv` - array of arguments as strings
  - application name counted as an argument, so `argc` is at least 1
- `int main()` is also valid if you don't care about command line arguments

Data Types 1/3 (Primitives)

- `int` - integer (size is platform dependent)
- `int32_t` - 32-bit integer on all platforms
- `float` - floating point number
- `char` - character
- `int[10]` - array of 10 integers
- `char[10]` - array of 10 characters (a string)
- ...

Data Types 2/3 (structs)

```
struct person
{
 char[20] name;
 int age;
 char[256] address;
};
```

- `struct` types hold collections of elements
- `struct person` (both words together) is now a "data type"
- Declare variables as such:
  `struct person john_doe;`
  - the '.' operator is used to access `struct` members
    `john_doe.age`

Data Types 3/3 (Pointers)

- Pointers are variables whose contents are interpreted as the memory addresses of other variables

Data Types 3/3 (Pointers)

- Pointers are variables whose contents are interpreted as the memory addresses of other variables
- `int *value;`
  `struct person *john_doe;`
- Operators relating to pointers
  - `*` - *dereference*: follow pointer and read data value
  - `&` - *address of*: get address of a variable (usually to store in pointer)
  - `->` - *access element*: access elements of a struct pointer. Equivalent to `(* _).`

Memory

- Stack
  - memory allocated statically by compiler
  - memory released back to system automatically after function returns
- Heap
  - memory allocated dynamically by programmer at run-time
  - memory has to be released back to system (freed) manually by programmer
  - use `malloc( size )` and `free( pointer )` to allocate and free memory

    -
      ```
      int *ptr = (int *) malloc(sizeof(int));
      free(ptr);
      ```

Frequently Used Libraries

- `stdio.h` - provides `printf`, `scanf`, `fgets` and other input/output functions
- `stdlib.h` - provides `malloc` and `free` functions
- `string.h` - provides `strcmp`, `strcpy` and other string manipulation functions
- `stdint.h` - provides `int32_t`, `uint32_t`, `int64_t`, `uint64_t` and other fixed size integers

Remember to include the correct library when using something provided by it!

Debugging

- Debugging is extremely valuable to determine what is going wrong with your program
- `GDB` is an interactive command line debugger
    - `break <function name or line number>` - sets a break point at a function def. or a line #
    - `print <variable name or expression>` - prints the value of a variable or an expression on program variables
    - `run <command line arguments>` - starts running the program with the given command line arguments
    - `help` - find help on more commands
- Compile your code with the –g flag for gcc to be able to debug the program with gdb

Terminal Input/Output

- Output to screen
  - `printf("Hi %d", 5);` - print formatted text
  - `puts("Hello World!");` - print a string
- Input from keyboard
  - `gets(buffer_array);` - read a single line from `stdin` into the `buffer_array`
  - `fgets(buf, 128, stdin);` - read a single line or at most 128 characters from `stdin` into `buf`
  - `scanf("%s %d", buf, &i);` - read from `stdin` and "parse" input according to given format

Many more variants that work with any stream type (for example good for file I/O)

File I/O

Opening and closing files

```
int fd;                   // File Descriptor
fd = open("/path/to/file", O_RDWR | O_CREAT);
close(fd);
```

Reading and Writing

```
char buf[4096]; int len;
len = read(fd, buf, 4096)
len = write(fd, buf, 4096);
```

WARNING: Size passed is only a **suggestion**. May read/write fewer than requested number of bytes. Return value is number of bytes actually read/written. MUST retry if not fully read/written.

Network I/O

Opening and closing network sockets

```
int sock;                          // File Descriptor
sock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
close(sock);
```

Internet Addresses

```
struct sockaddr_in addr;
addr.sin_family = AF_INET;
addr.sin_addr.s_addr = htonl(0x7F000001);
addr.sin_port = htons(8080);
```

Fill the address info manually or get the info automatically with `getaddrinfo()`.
See `man getaddrinfo`

Bitwise Operations

- Manipulate individual bits in a variable
- Useful for many things, one of which is serialization
- Operators
    - `a & b` - bitwise AND
    - `a | b` - bitwise OR
    - `a ^ b` - bitwise XOR
    - `~a` - bitwise one's complement
    - `a << b` - bitwise shift left
    - `a >> b` - bitwise shift right

Threads

Starting a thread

```
#include <pthread.h>
...
pthread_t id;
err = pthread_create(&id, NULL, entry_func, arg);
```

Body of a thread

```
void *entry_func(void *arg) {
      ...
```

Exiting current thread

```
      ...
      pthread_exit((void *)return_value);
}
```

Resources

- Dave's programming in C tutorials:
  http://www.cs.cf.ac.uk/Dave/C/CE.html
- Linux manual pages
  ```
  man pthread_create
  man stdlib.h
  ```
  - also available online at http://linux.die.net/
- Search online!
  Plenty of resources and tutorials online

Questions ?

Sample Interview Questions

1. Swap two arbitrary variables without using a third temporary variable.
2. A linked list can become corrupt if the last element's "next pointer" points back to some previous element rather than `NULL`. This can cause functions that traverse the list to loop for ever. Write a small C function to determine whether a given linked list is corrupt or not.