C Programming and Program #1 Helper

Prof Bill - Jan 2017
This is a Helper document for the C Programming Language and CSC 210 Program #1
Video List: wtkrieger.faculty.noctrl.edu/csc210-winter2017/Program1.pdf
The sections are:

A. Introduction
B. The Command Line
C. C Coding
D. My Program #1 Stuff

thanks...yow, bill

A. Introduction

The C programming language is 45 years old (gasp), and it's the language I used in my first real nerd job out of school (gasp gasp). It's still important today.

The **book**:

The C Programming Language, ANSI C
by Brian Kernighan and Dennis Ritchie
www.ime.usp.br/~pf/Kernighan-Ritchie/C-Programming-Ebook.pdf

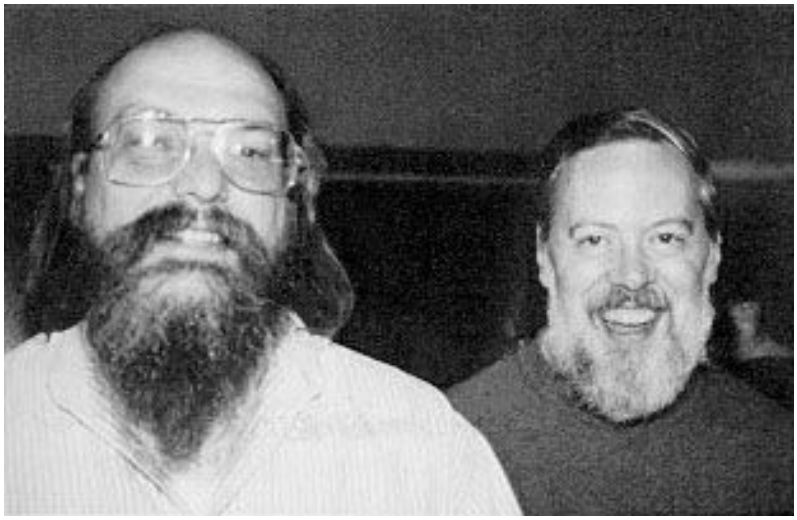
Update: I found a nicer PDF for the Kernighan and Ritchie C Programming Language book. You can search and jump in this one. Nice!

The C programming Language (PDF)

A little **history**: "The origin of C is closely tied to the development of the Unix operating system" and "Also in 1972, a large part of Unix was rewritten in C.[13] By 1973, with the addition of struct types, the C language had become powerful enough that most of the Unix's kernel was now in C."

You get a nice overview from Wikipedia:

https://en.wikipedia.org/wiki/C_(programming_language)



C is a lower-level programming language than Java. It doesn't have objects or classes or exceptions or interfaces or abstract thingies. A C program is a collection of functions. There's no garbage collection; you allocate and free memory yourself. C has pointers. We'll learn a lot about pointers in our programming assignment.

Fwiw, here's a big comparison table: introcs.cs.princeton.edu/java/faq/c2java.html
thanks… yow, bill

B. The Command Line

IDE's like Eclipse and NetBeans are great. Hey, so are windows and the mouse and GUI's and… But typing commands at a **command line** or shell is still an important skill for a programmer. Here are two ways to get started.

- On Windows, run **cmd**
- On Mac, run the **Terminal** application in Utilities

From the command line, file system from the command line:

- cd <folder> - change directory
- ls - list files

You can use Notepad++ to edit your C programs.

Compile C programs with the Gnu C compiler: gcc.gnu.org/

- The **gcc** compiler is available on our school computers. So is the debugger, **gdb**.
- If you have a Mac at home (like I do), gcc is included in Mac's Xcode library. I couldn't find gdd on my Mac, but (google google) a program called **lldb** was available and worked fine: lldb.llvm.org.

Here are some common gcc commands/options:

```
# compile hello.c, creating executable output a.out
gcc hello.c

# compile hello.c, creating executable output hello
gcc hello.c -o hello

# compile hello.c, with extra files for debugging
gcc hello.c -g -o hello
gdb hello

#compile hello.c to create hello.o, but no executable
gcc -c hello.c
```

Here's a nice, short summary of gdb debugging commands:
www.tutorialspoint.com/gnu_debugger/gdb_commands.htm

The gdb commands I use most are:

➔ l - list code, so you can see where to set breakpoints

- ➔ l <line-num> - list code at line number
- ➔ b <line-num> - set breakpoint at line-num
- ➔ run - run program to the next breakpoint
- ➔ n - next, execute one line
- ➔ s - step, execute one line of code, but step into function calls
- ➔ p <variable> - print value of a variable
- ➔ help
- ➔ q - quit

/* When I installed gcc on my MacBook, gdb was not there. (google google) So, on my Mac, I use lldb instead. The interface is nearly identical.

C. C Coding

There are a million online **tutorials** and other resources. I'll leave you to your googling. Of the stuff I've looked at, I think I liked this one best. We'll use this in class some.

> www.learn-c.org

Here are two different, easy summaries of functions in the C standard library:

> www.tutorialspoint.com/c_standard_library/index.htm
> en.wikipedia.org/wiki/C_standard_library

The most important of these are:

- ➢ stdio.h - always include this
- ➢ stdlib.h - you can always include this one too; includes malloc() and free()
- ➢ string.h - string functions

Naming conventions

The main thing here: be consistent. I chose this style from this SO post.

> The most important thing here is consistency. That said, I follow the GTK+ coding convention, which can be summarized as follows:
>
> 1. All macros and constants in caps: MAX_BUFFER_SIZE, TRACKING_ID_PREFIX.
> 2. Struct names and typedef's in camelcase: GtkWidget, TrackingOrder.
> 3. Functions that operate on structs: classic C style: gtk_widget_show(), tracking_order_process().
> 4. Pointers: nothing fancy here: GtkWidget *foo, TrackingOrder *bar.
> 5. Global variables: just don't use global variables. They are evil.
> 6. Functions that are there, but shouldn't be called directly, or have obscure uses, or whatever: one or more underscores at the beginning: _refrobnicate_data_tables(), _destroy_cache().

For example, in Program #1 I have a VideoList struct. I have functions like read_video_file(). And a constant like VIDEOS_FILE_NAME.

Pretend Objects

C isn't object-oriented. At all. But we can pretend… well, sort of.

Let's pretend to create an object (a "class" in Java) called Professor. A professor is a struct. We'll define the professor-related struct, a typedef name, and function definitions in one header file: **professor.h**.

```
#ifndef PROFESSOR_H
#define PROFESSOR_H

/* Part 1. Define the professor struct */
struct Professor {
      char * name;
      int dept_code;
      struct College *employer;
};
typedef struct Professor *Professor;  /* Part 2. a nicer name */

/* Part 3. extern the prof-related functions */
extern Professor *new_professor( char *nm, int code, struct
college *emp);
extern void grade_programs( Professor *p, ProgramList
*programs)
extern int get_tenure( Professor *p)

#endif
```

The **#ifndef, #define and #endif statements** are there to guard the header file. They prevent the header file from being compiled more than once.

Part 1. The professor **struct** and its fields are defined here.

Part 2. A **typedef** makes it a little nicer to reference a professor as "Professor", rather than "struct professor".

Part 3. The **extern** allows functions in other files to call your Professor functions.

Notice the big difference here between C and Java. This object setup is all optional in C. In Java, it's all part of the language.

Some K&R notes

Here are some supplemental notes for the K&R C book:

The C programming Language (PDF)

Ch 1 Tutorial Intro

- C printf is like printf in Java
- Use #define for constants; #define MAX_LINE 100
- All function arguments are pass-by-value. You can use the “address of” operator (&) to pass the address of a variable into a function. This is also called a “pointer to” the variable. See swap() function example in Chapter 5!
- In C, char array (char []) or char pointer (char *) is used to represent a string. You must allocate space for the string characters if using char *. Unlike JAva, these C strings are mutable.

Ch 2 Types, Operator, Expressions

- Only 4 basic built-in types in C: char, int, float, double

Ch 3 Control Flow

Ch 4 Functions

Ch 5 Pointers and Arrays

- See swap() function for an excellent example of pointers and the address of operator (&). Use & to effectively pass-by-reference.
- Pointers and arrays are very similar. char * is like char [].
- In C, you can define a pointer to a function. Cool. We won’t need this though.

Ch 6 Structures

- A struct defines related fields, like a class in Java. No methods though!

```
struct Person {
  char *name;
  int dept_code;
  double hourly_salary;
};
```

- Access a field for struct with dot (.): p.name. Access field for a struct pointer using two-char arrow (->): p2->name. We usually deal with pointers to structs!
- structs can be self-referential, as in list nodes.
- typedef struct XXX XXX… to rename the struct and make code a little cleaner.

Ch 7 Input Output

- #include <stdio.h>... I always do this too: #include <stdlib.h>
- you can redirect stdin with < in command line. Redirect stdout with >.

    - program1 < test1.txt # test.txt is now stdin
    - program1 < test1.txt > test1_out.txt
- File access. Google fopen().
    - FILE *fp;
    - fp = fopen(“test.txt”, “r”);
- In Unix, programs return integers. exit(0) on success. exit(1), or any non-zero value on error.
- Use malloc or calloc for memory allocation. See examples. On diff: calloc zeroes out memory that has been allocated. Use free() to free up space.
    - int array1[100]; // static array
    - int *array2; // dynamic array
    - ip = (int *) calloc(100, sizeof(int));

Ch 8 Unix

D. My Program #1 stuff

I'll just rattle off some stuff here from my implementation. Some will apply to you. Some won't. Also, I'm still figuring out Piazza, so we'll see how that fits in as well. Ideally, we can post code snippets in Piazza to share. We'll see.

C functions and other stuff I used in my solution:

- I used **enum** for my commands. This was just showing off.
- **File stuff** I used to read and save my videos.txt file: FILE type, fopen(), fclose(), fgets(), fputs()
- **fgets()** preserves the newline in the input string. I got rid of this with:
  - `line[ strlen( line) - 1] = '\0';`
- Speaking of, **string functions** I used: strlen(); strncmp() - to compare the first N chars of a string
- I used **scanf()** to read an integer for my select number:
  - `int value = -1;`
  - `scanf( "%d", &value);`
- **getline()** worked at home, but not here at school. I replaced it with **fgets()**.
- Declaring a variable in a **for loop** [for(int i = 0;…] worked at home and not at school. I just pulled the declaration out of the loop.
- I usually **include** <stdio.h> and <stdlib.h> in every C file. Then, I'll add extra system headers as I need them.
- I used **rand()** and **srand()** for adding to a random position in the list. If you want to seed with the current time:
  - #include <time.h>
  - ...
  - srand((unsigned) time(&t));
- Use **malloc()** to allocate memory (for a struct like Node or VideoList).
- Don't forget to **free()** when you remove a video!
- I copied the video name when storing it in the Node. Weird things can happen if a string gets reused.
  - char *s = (char *) malloc(strlen(data) * sizeof(char));
  - strcpy(s, data);

Go!

- I like two character commands. I'm only looking at the first 2 chars of commands as they come in. pr for print. ad for add. And so on.

    - Darn it. I implemented reverse, and it conflicts with remove. Switch command to delete?
- I added a help command which lists all my commands.
- I added a verbose command that turns on/off printing the Video List every time.
- My Node struct is defined in my video_list.c file because it's "private".
- Use gdb to debug your list code. If you have a Node *n, then print n shows you the value of n, which is a point. Now, try print *n. This shows you all the fields of n. Very handy!
-

thanks... yow, bill