ECE 264 Advanced C Programming

If you want to try the code in the handout (PDF file), you do not have to type it. In Adobe Reader, select `File` and `Save as Text`.

Course Web Site: `https://engineering.purdue.edu/OOSD/S2010`

Contents

1 What is C?

C was developed in early 1970s in Bell Lab for programming the UNIX operating system. At that time, OS was written primarily in Assembly. From the beginning, high performance is a major goal in C. C is one of the mostly widely used programming languages now. You can find C in high performance supercomputers as well as tiny embedded systems. In fact, there are many variations of C. Some applications extend the features of C and some others restrict C. C is different from some other types of languages, especially script languages. In C, you need to declare a variable before using it. This is different from MATLAB and script programming. You have learned MATLAB in CS 159 and you will learn script programming in ECE 364. A brief history of programming languages is available at

`http://oreilly.com/news/graphics/prog_lang_poster.pdf`.

Computer programs are everywhere in our daily lives. Think of Google, Facebook, Amazon... cellular phone, text messaging ... What are typical functionalities of a program?

- Iterations: When you search a book, the database goes through, i.e. *iterates*, many books to check whether any book matches your keyword (or keywords).
- Comparison / Condition: This "checking" compares your keyword and the keywords associated with the books.
- Collection of Information: The list of books matching your search is created and displayed.

The program also has to handle networking and formatting an HTML page that is shown on your browser. In fact, there are probably several programs, one for receiving your request, one for searching the database, and one for creating and formatting the output.

2 Type System

C has a *type system* to prevent programmers from making **"obvious"** mistakes (it cannot prevent programmers from making "non-obvious" mistakes). For example, you can write

```
/* type1.c */
/* This is a comment. */
int a = 4; /* a is an integer.  */
int b = 7;
int c = a + b; /* c is 11.  */
```

You can also mix types when the mixing makes sense:

```
/* type2.c */
int d = 2;
float e = 11.05; /* single-precision floating point number */
float f = d + e; /* f is 13.05 */
```

However, you may not mix types when it makes no sense. For example

```
/* type3.c */
Book bo;
/* Book is a programmer-defined type */
Bike bi;
/* Bike is another programmer-defined type */

int g = bo + bi;
/* What does this mean? */
```

You need three steps to execute a C program: *compile*, *link*, and *execute*. A compiler translates a C source program (a text file we can read) to an *object file* (.o). Linker takes several object files and creates an executable. Finally, the program is executed. C checks types at compilation to prevent run-time type errors. We will learn more about this concept later.

Assume that you will write and rewrite (and rewrite and rewrite) the same program many times (because you **will**). Do no assume that your program is perfect after you finish typing. Assume you are going to change the program many times. Make your program clear and easy to read (variable names, indentation ...)

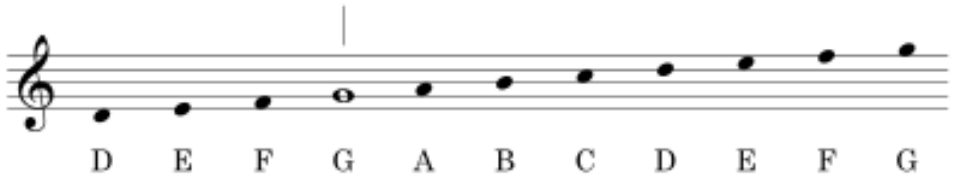
3 First Example

```
/* first.c */
#include <stdio.h>
int main(int argc, char * argv[])
{
  int cnt;
  for (cnt = 0; cnt < argc; cnt ++)
    {
      printf("%s\n", argv[cnt]);
    }
  return 0;
}
```

The first line is a comment. The second line tells a C compiler (in fact, it is the "preprocessor" but we can ignore the difference for the time being) to include the standard IO library. Why do we need to do this? Because we are going to use a function that will output some information. We will talk about this in just a moment. The program has a function called `main`. A C program usually has several functions. The `main` function is special because it is the initial point of the program. This function returns an integer value and takes two input arguments. The first argument has the type of an integer. The second is an array of character arrays. C does not have strings; instead, each string is an array of characters (we will talk about arrays later). We usually call the first argument `argc` and the second `argv` to represent the "counter" of arguments and the "values" of arguments. Each function is implemented by the code inside a pair of curly brackets `{ .... }`. Inside the function body, we declare an integer variable called `cnt` to mean a counter. This counter iterates through the input arguments and prints one argument on each line. This `printf` function is a function provided in the standard C library. That is the reason we need to include `stdio.h` at the top. It means the standard C library for input (such as from a keyboard) and output (such as printing to a screen). This is called a *header file*. Finally, the program returns 0. In C, if a function completes successfully, it usually returns

0. If the function fails, it usually returns -1. In ECE 264, the return value of `main` may not seem important. When you do script programming in ECE 364, you may need to check whether a program terminates successfully. In that scenario, the return value becomes important.

You probably have many questions about why C is written in this way. A programming language has many rules and also many conventions. These rules are different from the rules of physics. A programming language is invented by some people and they decide the language should have these rules. They certainly have their own reasons. A programming language is similar to a natural language, such as English or Chinese. Why do you put a subject before a verb in English? Meanwhile, a programming language also has convention. For example, the input arguments of `main` are called `argc` and `argv`. Can you change them? Yes. You can call them `dog` and `bird` and the program does exactly the same thing. C has a collection of words that are reserved for special meanings. These are called *keywords* (page 77 in ABOC). As long as you do not use the keywords, the compiler does not care whether you call it `argc` or `dog`. However, you have to be careful when you violate the convention. Your program will be more difficult to read, possibly by yourself later. Rules and convention can be good. Consider this example



Everyone knows exactly what you mean when you put a note at a particular location. These are the rules. The rules make communication between musicians easier. Composers also follow certain convention when they write music. For example, a symphony usually has four movements. Do these rules restrict the creativity of musicians? I don't think so. Can you say Mozart not creative? When you write a program, you have to follow certain rules and convention. Your creativity is built upon these rules and convention so that everyone knows exactly what you mean.

4 Compile and Execute C Program

How do we compile, link, and execute the program? Let's call the program ex1.c.

```
gcc -Wall -Wshadow ex1.c -o ex1
```

This program has only one file so we can compile and link in one step using `gcc`. It is a C compiler available for many types of computers. It is open-source and free of charge. The *flag* `-Wall` means to turn on all warning messages. You should use this because **warnings usually indicate errors**. Another flag is `-Wshadow`: it informs you when two variables of the same name overlap in their scopes. This is likely an error. The file name `ex1.c` is provided and the output is `ex1`. If you do not give `-o` and the output name, the default output is `a.out`. To execute this program, simple type

```
./ex1
```

here `./` means the current directory (also called *folder*). This is important because it is possible to have another program called `ex1` somewhere else on your computer. You want to execute the program at this directory. **A common error among beginning programmers is to call their program as "test". It turns out "test" is also a program provided by UNIX.** If you do not use `./test`, you will call the wrong one (usually in `/usr/bin/test`).

If we execute the program by typing

```
./ex1 ece 264 spring 2010
```

the output is

```
./ex1
ece
264
spring
2010
```

The program prints the arguments one-by-one, as promised, starting from the program's name `./ex1`.

5 Second Example

This example shows how to create a function and how to call it.

```
/* second.c */
#include <stdio.h>
#include <stdlib.h>
int add(int a, int b)
{
  return (a + b);
}

int main(int argc, char * argv[])
{
  int val1;
  int val2;
  if (argc < 3)
    {
      fprintf(stderr, "need two numbers\n");
      return -1;
    }
  val1 = (int)strtol(argv[1], (char **)NULL, 10);
  val2 = (int)strtol(argv[2], (char **)NULL, 10);
  printf("%d + %d = %d\n", val1, val2, add(val1, val2));
  return 0;
}
```

We also include another header file called `stdlib.h` for standard C library. This program creates a function called `add`. It takes two integer arguments and returns the sum of the two arguments. The `main` function requires three arguments: the first one is the program's name (this is always available), the second and the third are two numbers. If the program is not given the three arguments, the program prints an error message and returns -1. **When the `main` function returns, this program terminates.** If there are three arguments, `val1` and `val2` are obtained by converting the arguments from characters to integers. This is achieved by calling the `strtol` function. This function is provided by the standard C library. Next, we are going to print the values of val1, val2, and the sum of the two values. Let's call this program ex2 and execute it.

```
./ex2 4 9
```

The output is

```
4 + 9 = 13
```