CS429: Computer Organization and Architecture

Intro to C

Warren Hunt, Jr. and Bill Young
Department of Computer Sciences
University of Texas at Austin

Last updated: September 3, 2014 at 08:38

Topics

- Simple C programs: basic structure, functions, separate files
- Compilation: phases, options
- Assembler: GNU style, byte ordering, code and data segments
- Tools for inspecting binary: od, objdump

A Simple C Program

- A first program is to just print a short message.
- We assume our target is a 32-bit, x86-compatible machine.
- This program prints “Hello, world!” to its standard output.
- We use gcc to compile the program.

```
/* Hello, world! Program */

#include "stdio.h"

int main()
{
   printf("Hello, world!\n");
}
```

Running the Program

Several steps are necessary to run the program.

- Invoke the `gcc` *compiler driver* to transform your text file (in this case called `hello.c`) into an executable image.
- Then ask the *operating system* to run the executable.

```
> gcc hello.c
> a.out
hello, world
>
```

A More Complex Program

```
#include <stdio.h>

/* print Fahrenheit to Celsius table
   for fahr = 0, 20, ..., 300 */

main()
{
  int fahr, celsius;
  int lower, upper, step;

  lower = 0;      /* low limit of table */
  upper = 300;    /* high limit of table */
  step = 20;      /* step size */
  fahr = lower;
  while (fahr <= upper) {
    celsius = 5 * (fahr-32) / 9;
    printf("%d\t%d\n", fahr, celsius);
    fahr = fahr + step;
  }
}
```

Running the Temperature Program

```
felix:~/cs429/c> gcc -O2 temperature.c
felix:~/cs429/c> a.out
0 -17
20 -6
40 4
60 15
80 26
100 37
120 48
140 60
160 71
180 82
200 93
220 104
240 115
260 126
280 137
300 148
```

Specifying an Output Filename

```
felix:~/cs429/c> gcc -O2 -o tempConvert temperature.c
felix:~/cs429/c> tempConvert
0 -17
20 -6
40 4
60 15
80 26
100 37
120 48
140 60
160 71
180 82
200 93
220 104
240 115
260 126
280 137
300 148
```

TempConvert with For Loop

```c
#include <stdio.h>

#define  LOWER 0    /* low limit of table */
#define  UPPER 300  /* high limit of table */
#define  STEP  20   /* step size */

/* print Fahrenheit to Celsius table
   for fahr = 0, 20, ..., 300 */

main()
{
  int fahr;
  double celsius;

  for (fahr = LOWER; fahr <= UPPER; fahr += STEP) {
    celsius = (5.0 / 9.0) * (fahr - 32);
    printf("%3d %6.1f\n", fahr, celsius);
  }
}
```

Running TempConvert2

```
felix:~/cs429/c> gcc -o tempConvert2 temp2.c
felix:~/cs429/c> tempConvert2
  0  -17.8
 20   -6.7
 40    4.4
 60   15.6
 80   26.7
100   37.8
120   48.9
140   60.0
160   71.1
180   82.2
200   93.3
220  104.4
240  115.6
260  126.7
280  137.8
300  148.9
```

Program with Environment Variables

- This program has environment input variables.
- Variables argc and argv reflect the command line.
- Variable env reflects the environment variables.

```
#include ”stdio.h”        // for the printf command

main( int argc, char *argv[], char *env[])
{
    printf(”Status: number of command line args.\n”);
}
```

Note that the env parameter is not in the standard, but is widely supported.

Command Line Arguments

```c
#include "stdio.h"

main( int argc, char *argv[], char *env[])
{
    int i;
    if( argc == 1 )
        printf( "The command line argument is:\n" );
    else
        printf( "The %d command line arguments are:\n",
            argc );

    for( i = 0; i < argc; i++ )
        printf( "Arg %3d: %s\n", i, argv[i] );
}
```

argc is the argument count, including the name of the program.

argv is an array of those strings.

Running the Program

Here's a compilation and run of the program:

```
> gcc −o commargs commargs.c
> commargs x y z 3
The 5 command line arguments are:
Arg   0: commargs
Arg   1: x
Arg   2: y
Arg   3: z
Arg   4: 3
```

Command Line Arguments

env holds an array of strings maintained by the OS.

```
#include "stdio.h"
#include "stdlib.h"

main( int argc, char *argv[], char *env[])
{
    int i;
    printf( "The environment strings are:\n" );

    i = 0;
    while( env[i] != NULL )
    {
        printf( "Arg %3d: %s\n", i, env[i] );
        i++;
    }
}
```

Running the Program

```
> gcc -o envargs envargs.c
> envargs
The environment strings are:
Arg  0: PWD=/u/byoung/cs429/c
Arg  1: TERM=dumb
Arg  2: TERMCAP=
Arg  3: COLUMNS=80
Arg  4: EMACS=t
Arg  5: INSIDE_EMACS=23.3.1,comint
Arg  6: SHELL=/lusr/bin/tcsh
Arg  7: GROUP=prof
Arg  8: GPG_AGENT_INFO=/tmp/keyring-hZHfuV/gpg:0:1
# <lots more, 49 in all>
```

The GNU GCC Compiler

gcc is a cross compiler

- It runs on many machines
- Input languages: C, C++, Fortran, Java, and others
- Many target languages: x86, PowerPC, ARM, MC680x0, others

Extensive documentation is available on-line.

gcc works in phases:

```
gcc -v -O2 -o <objectFile> <sourceFile>.c
```

GCC can be used to print assembler:

```
gcc -S -O2 <sourceFile>.c
```

Assembler Output from gcc

You can produce assembler output, without running the assembler.

```c
int sum( int x, int y)
{
    int t = x + y;
    return t;
}
```

To generate the assembler in file `sum.s`:

```
gcc -S -O2 -c sum.c
```

sum.s

```
                .file ”sum.c”
                .text
                .p2align 4,,15
.globl sum
                .type      sum, @function
sum:
                pushl      %ebp
                movl       %esp, %ebp
                movl       12(%ebp), %eax
                addl       8(%ebp), %eax
                popl       %ebp
                ret
```

Assembler Output from Binary

objdump can be used to disassemble binary output.

```
00000000  <sum>:
 0:          55                  push      %ebp
 1:          89 e5               mov       %esp, %epb
 3:          8b 45 0c            mov       0xc(%ebp), %eax
 6:          03 45 08            add       0x8(%ebp), %eax
 9:          5d                  pop       %ebp
 a:          c3                  ret
```

Show Bytes Program

```
#include <stdio.h>
typedef unsigned char *byte_pointer;

void show_bytes(byte_pointer start, int len) {
  int i;
  for ( i = 0; i < len; i++ ) {
    printf("%.2x", start[i]); }
  printf("\n");
}

void main (int argc, char *argv[], char *env[] ) {
  int i = 15213;
  float f = 15213.0;
  double d = 15213.0;
  int *p = &i;

  show_bytes((byte_pointer) &i, sizeof(i));
  show_bytes((byte_pointer) &f, sizeof(f));
  show_bytes((byte_pointer) &d, sizeof(d));
  show_bytes((byte_pointer) &p, sizeof(p));
}
```

Running show_bytes

Here's how you might compile and run that code:

```
> gcc -o showbytes showbytes.c
> showbytes
 6d 3b 00 00
 00 b4 6d 46
 00 00 00 00 80 b6 cd 40
 f4 88 f2 bf
```

C Tutorials Available

Google “C tutorial” and you’ll find lots of options. For example: `http://wwww.iu.hio.no/~mark/CTutorial/CTutorial.html`

The C Programming Language, 2nd edition, by Kernighan and Richie is a standard reference. There are versions available on-line.