C Tutorial

Session #5

- File I/O
- Bit Masks, Fields & Bit Manipulations

Communicating with files

- Often times, you need read information from a file or write information into a file.
- What is a file? Or what is a file to your C program.
- C views a file as a continuous sequence of bytes, each of which can be read individually.

The text view and the binary view

- The two views of a file are binary and text.
- In binary view, each and every byte of the file is accessible to a program.
- In text view, the local environment's representation are mapped to the C view when a file is read.

Standard files

- C programs automatically open three files on your behalf, *standard input, standard output* and *standard error output*.
- *Standard input*, by default is your normal input device, usually your keyboard.
- Both *standard output* and *standard error output*, by default, are your normal output device, usually your display screen.

The *fopen* function

- Syntax: *fopen(char *filename, char *mode)*
- Example:

```
FILE *fp;
if ((fp = fopen("filename","r"))== NULL) {
    printf("can't open file.\n");
    …
}
```

Checking command-line arguments

- Pass arguments from shell command.
- Example:

  *int main(int argc, char *argv[])*

- The first argument *argc* is the number of arguments passed to your program.
- The array *argv* strores arguments. a*rgv[0]* is the program name, *argv[n]* is the n*th* argument passed to your program.

The *getc* and *putc* function

- Similar to getchar function, getc reads one byte from the target file.
- In our previous example, we can use *getc* as following:

  *FILE *fp;*

  …

  getc(fp);

- *getc* returns an EOF symbol when reaches the end of the file.

Other file I/O functions

- *fprintf(FILE *, format, args..)*
- *fscanf(FILE *, format, args…)*
- *fgets(char *buf, int MAX, FILE *)*
- *fputs(char *buf, FILE*)*
- *fclose(FILE *)*

The *fseek function*

- The fseek function enables you to treat a file like an array.
- Syntax: *int fseek(FILE *, long offset, int whence);*
  - *fseek(fp, 0L, SEEK_SET); //go to the beginning of the file*
  - *fseek(fp, 10L, SEEK_SET); //go 10 bytes into the file.*
  - *fseek(fp, 2L, SEEK_CUR); //advance 2 bytes from the current position.*
  - *fseek(fp, -10L, SEEK_END); //back up 10 bytes from the end of the file.*

The *ftell* and *rewind* functions

- The *ftell* function returns the current value of the file position indicator.
- The *rewind* function sets the file position indicator to the beginning of the file stream.
- The *rewind* function is equivalent to:

  fseek(fp, 0L, SEEK_SET);

Number Notation

- Decimal - 10
- Unsigned – 10u
- Long – 10L
- Floating Point 10.0
- Octal (Base 8) – 010 (8 decimal)
- Hexadecimal (Base 16) – 0x10 (decimal 16)

Bit fiddling

- With C, you can manipulate the individual bits in a variable.
- Why we need to play with bits?
  - Hardware devices if often controlled by sending it one byte or two, in which each bit has a particular meaning.
  - Many compression and encryption operations manipulate individual bits.

C's bitwise operators

- & AND
- | OR
- ^ XOR
- ~ One's Complement (unary)
- << Left shift
- >> Right Shift

DO NOT confuse & with &&

& is bitwise AND, && logical AND

Similarly for | and ||

Turn on 1 bit

- We can use bitwise or to turn on a bit flag.
- Sample code:

```
int flag;
flag = flag | 8;  // turn on the forth bit from right.
```

Note 8 in binary is 00001000

- Similarly if we want to turn on the 1st and 3rd bits, we can use *flag = flag | 5;*

Turn off 1 bit

- We can use bitwise and to turn off 1 bit.
- Sample code:

  int flag;

  flag = flag & (~4) // turn off the 3rd bit from right.

- Similarly, if we want to turn off 1st, 2nd, and 4th bits from right, we can use flag = flag & (~11);

Bit shifting

The shift operators perform appropriate shift by operator on the right to the operator on the left. The right operator must be positive. The vacated bits are filled with zero (i.e. There is NO wrap around).

Example:

```
x = 0x02;    // 00000010
x >>= 2;     // 00000000

x = 0x02;    // 00000010
x <<= 2;     // 00001000
```

Therefore a shift left is equivalent to a multiplication by 2

Similarly a shift right is equal to division by 2

Number << n multiplies number by 2 to the nth power.

Number >> n divides number by 2 to the nth power.

Bit fields

```
Struct box_props{
    unsigned int opaque          :1;
    unsigned int fill_color      :3;
    unsigned int show_border     :1
    unsigned int border_color    :3
    unsigned int border_style    :2
}
```

Examples

- Consider the following mask, and two bit strings from which we want to extract the final bit:

```
mask = 0x01;   // 00000001
value1 = 0x9B; // 10011011
value2 = 0x9C; // 10011100
x = mask & value1; // 0x01 (00000001)
x = mask & value2; // 0x00 (00000000)
```

- The zeros in the mask *mask off* the first seven bits and only let the last bit show through. (In the case of the first value, the last bit is 1; in the case of the second value, the last bit is 0.)
- Alternatively, masks can be built up by operating on several flags, usually with inclusive OR:

```
flag1 = 00000001;
flag2 = 00000010;
flag3 = 00000100;
mask = flag1 | flag2 | flag3; // mask = 00000111 (0x07)
```