# Agenda

Bitwise Operators: and, or, xor, not

Left/right shifts

Bitfields

Standard I/O

File I/O

- File pointers
- Text files: sscanf, strtok, fprintf, fscanf
- Binary files: fread, fwrite

# Bitwise operations: AND, OR, XOR, NOT

Ex: 4 & 8
Ex: 4 | 8
Ex: ~4

*Table 1. The Results of Bitwise ANDing Two Values (A AND B)*

| A | B | A & B |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

*Table 2. The Results of Bitwise ORing Two Values (A OR B)*

| A | B | A \| B |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

*Table 3. The Results of Bitwise XORing Two Values (A XOR B)*

| A | B | A ^ B |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

*Table 4. The Results of Bitwise NOTing a Value (A)*

~A

| A | ~ A |
|---|---|
| 0 | 1 |
| 1 | 0 |

# Exercise: Compute a & b

Give your final result as a hexadecimal number

Suppose a = 7 and b = 4.

# Exercise: Compute a | b

Suppose a = 7 and b = 4.

# Exercise: Compute a ^ b

Suppose a = 7 and b = 4.

# Exercise: Compute ~a

Suppose a = 7 and b = 4.

# Bitops: Code example

```
#include <stdio.h>

int main() {
  unsigned char a = 7;
  unsigned char b = 4;
  unsigned char not = ~a;

  printf("%02X & %02X = %02X\n", a, b, a & b);
  printf("%02X | %02X = %02X\n", a, b, a | b);
  printf("%02X ^ %02X = %02X\n", a, b, a ^ b);
  printf("~%02X = %02X\n", a, not);
  printf("%lu\n", sizeof(unsigned char));

  return 0;
}
```

# Bit shift

<< shifts to the left

>> shifts to the right

When shifting to the left, fill in with zeros

When shifting to the right,

for signed integers: fill with most significant bit (**logical shift**)

for unsigned integers: fill with zero (**arithmetic shift**)

# Exercise: bit shifts

```
unsigned char left = 0xAA00;
unsigned char shift1 = left >> 8;
unsigned char shift2 = left >> 3;
unsigned char shift3 = left << 4;
unsigned char shift3 = left << A;
```

# Exercise: bit shifts

```
char left = 0xAA00;
char shift1 = left >> 8;
char shift2 = left >> 3;
char shift3 = left << 4;
char shift3 = left << A;
```

# Bit shift: example

```
#include <stdio.h>

int main(int argc, char **argv) {
   /* Unsigned integer value: u_val. */
   unsigned int u_val = 0xFF000000;

   /* Signed integer value: s_val. */
   int s_val = 0xFF000000;

   printf("%08X\n", u_val >> 12);  // logical right shift
   printf("%08X\n", s_val >> 12);  // arithmetic right shift

   return 0;
}
```

# Example: Test for even/odd numbers

What value should we use as a mask?

(Assume 8 bit integers)

# Example: Use a mask to test if two signed numbers have opposite signs

What value should we use as a mask?

(Assume 8 bit integers)

# Demo: Swapping bytes

Suppose a = 0xAABB. Write a program that swaps the lowest 2 bytes.

```
unsigned int a = 0xAABB;
 unsigned int leftMask = 0xFF00;
 unsigned int rightMask = 0x00FF;
 unsigned int left = (leftMask & a) ;
 unsigned int right = (rightMask & a) ;
 unsigned int leftShift = left >> 8;
 unsigned int rightShift = right << 8;
 unsigned flipped = leftShift | rightShift;

 printf("Left: %08X Right: %08X\n", left, right);
 printf("Left: %08X Right: %08X\n", leftShift, rightShift);
 printf("Before: %08X After: %08X\n", a, flipped);
```

# Exercise: Swapping bytes

| Variable | Bits 24-32 | Bits 16-23 | Bits 8-15 | Bits 0-7 |
|---|---|---|---|---|
| a | 0000 0000 | 0000 0000 | 1010 1010 | 1011 1011 |
| leftMask | | | | |
| rightMask | | | | |
| Left | | | | |
| Right | | | | |
| leftShift | | | | |
| rightShift | | | | |
| flipped | | | | |

# Bit fields, flags, and masks

A **bit field** stores a set of booleans within a single value

A **flag** is a value that can be true or false

A **mask** is a value that zeroes out bits

Example: Suppose we store 8 flags within a character data type

```
char c = 0x45; // binary is 0100 0101
               // all flags are set to zero except for 3
```

# Example: Dinner options

```
#define WATER 0x01 // mask that corresponds to the least significant bit
#define WINE 0x02 //mask corresponds to the second least significant bit
#define DINNER_ROLL
#define SALAD
#define SOUP
#define MAIN
#define DESSERT
#define COFFEE

// Fill in the remaining masks
```

# Example: Dinner options

What is the bit field corresponding to each of the following options?

| Options | Bit Field |
| --- | --- |
| WATER, WINE, SALAD, MAIN, COFFEE | |
| WATER, SALAD, SOUP, MAIN, DESSERT | |
| WINE, MAIN, DESSERT, COFFEE | |
| WATER, SALAD, DESSERT | |

# Example: Dinner options

What options correspond to the following bit fields?

| Bit Field | Options |
| --- | --- |
| 0xAE | |
| 0x31 | |
| 0x9B | |
| 0x74 | |

# Example: File permissions use bit fields

```
alinen@sutekh:~/cs355/os-devel$ ls -l
-rw-r--r--  1 alinen alinen   70 Dec 10  2023 README.md
```

What numeric value corresponds to the following permissions (in octal)? –rw-r--r--

What numeric value corresponds to the following permissions (in octal)? –rwxr-x--x

# Standard I/O

Three default streams

- stdin (scanf, fscanf)
- stdout (printf, fprintf)
- stderr (fprintf(stderr, <format string>, <data>))

**stdin**, **stdout**, **stderr** are tied to files with descriptors 0, 1, and 2 respectively

At the command line we can redirect these streams to files

- ./a.out < infile.txt
- ./a.out > outfile.txt
- ./a.out &> allfile.txt (all output)
- ./a.out 2> errfile.txt (error output only)
- ./a.out 1> outfile.txt is the same as ./a.out > outfile.txt

# Standard I/O (C)

Writing/reading to terminal is equivalent to writing/reading from special files called stdout and stdin

```
int a;
 char word[32];
 printf("Enter an integer: ");
 scanf(" %d", &a);

 printf("Enter a string: ");
 scanf(" %s", word);

 printf("\nYou entered: %d %s\n", a, word);
 fprintf(stderr, "Test printing an error\n");
```

```
int a;
 char word[32];
 fprintf(stdout, "Enter an integer: ");
 fscanf(stdin, " %d", &a);

 fprintf(stdout, "Enter a string: ");
 fscanf(stdin, " %s", word);

 fprintf(stdout, "\nYou entered: %d %s\n", a, word);
 fprintf(stderr, "Test printing an error\n");
```

These two code examples are equivalent

# File I/O

Reading/writing files is done using a **file pointer.**

The **file pointer** keeps track of your current location within the file

**Text files** store character data

**Binary files** store data types in their native binary formats

```
FILE *infile; FILE *outfile;

infile = fopen("input.txt", "r");
if (infile == NULL) {
   printf("Error: unable to open file %s\n", "input.txt");
   exit(1);
}

outfile = fopen("output.txt", "w");
if (outfile == NULL) {
   printf("Error: unable to open outfile\n");
   exit(1);
}

fclose(infile);
fclose(outfile);
```

Text

```
FILE *infile; FILE *outfile;

infile = fopen("input.txt", "rb");
if (infile == NULL) {
   printf("Error: unable to open file %s\n", "input.txt");
   exit(1);
}

outfile = fopen("output.txt", "wb");
if (outfile == NULL) {
   printf("Error: unable to open outfile\n");
   exit(1);
}

fclose(infile);
fclose(outfile);
```

Binary

# Example: Reading Text Files (C )

```
const char* filename = "grades.txt";
FILE* fp = fopen(filename, "r");
if (!fp) // fp is NULL on error
{
  printf("Cannot load file: %s\n", filename);
  return 1;
}
int num = 0;
float sum = 0;
char line[1024];
while (fgets(line, 1024, fp))
{
  int grade;
  sscanf(line, " %d", &grade);
  sum += grade;
  num++;
}
printf("The average is %.2f\n", sum/num);
fclose(fp);
```

```
99
79
51
92
56
89
63
```

# Helpful text file commands

- fprintf, fscanf
- fgets, fputs (gets/puts a line of input)
- fgetc, fputs (gets/puts a single character)

# Parsing text using strtok

strtok tokenizes a line of text

```
void print(char** list, int n) {
  for (int i = 0; i < n; i++)   {
    printf("%s, ", list[i]);
  }
  printf("\n");
}
```

```
int main() {
    char line[1024];
    strcpy(line, "cat bear dog");

    int n = 0;
    char* list[16]; // max of 16 words can be put in the list
    char* token = strtok(line, delims);
    while (token && n < 16)     {
        list[n] = token;
        token = strtok(NULL, delims);
        n++;
    }
    print(list, n);
}
```

Draw the stack on the following slide

# Parsing using strok

# Aside: Viewing binary files: hexedit

```
00000000  00 00 00 00  00 00 00 00  01 00 00 00  02 00 00 00  02 00 00 00  ................
00000014  04 00 00 00                                        ....
```

# Demo: Writing Binary Files

```
#include <stdio.h>
struct Data {
  int size;
  float p[3];
  char buff[16];
};

int main()
{
  struct Data data = {10, 1.0, -2.0, 0.5, "carrot"};
  FILE* fp = fopen("data.bin", "wb");
  fwrite(&data, sizeof(struct Data), 1, fp);
  fclose(fp);
}
```

Hexedit Output

```
00000000   0A 00 00 00  00 00 80 3F  00 00 00 C0  00 00 00 3F  .......?.......?
00000010   63 61 72 72  6F 74 00 00  00 00 00 00  00 00 00 00  carrot..........
00000020
```

# Demo: Reading Binary Files

```
struct Data data;
 FILE* fp = fopen("data.bin", "rb");
 fread(&data, sizeof(struct Data), 1, fp);
 fclose(fp);
```

Hexedit Output

```
00000000   0A 00 00 00  00 00 80 3F  00 00 00 C0  00 00 00 3F  .......?.......?
00000010   63 61 72 72  6F 74 00 00  00 00 00 00  00 00 00 00  carrot..........
00000020
```

# Helpful binary file commands

FILE* fp = fopen(filename, “rb”);

// Write binary data

size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream)

// Read binary data

size_t fread(const void *ptr, size_t size, size_t nmemb, FILE *stream)

# Demo: Writing binary files

Draw the contents of the binary file

```
struct point {
  int x;
  int y;
};

int main() {
  struct point p[3];
  for (int i = 0; i < 3; i++) {
    p[i].x = i;
    p[i].y = 2*i;
  }

  FILE* fp = fopen("points.bin", "wb");
  fwrite(p, sizeof(struct point), 3, fp);
  fclose(fp);
}
```

# Demo: Comparing binary to text files

```
struct point {
  int x;
  int y;
};

int main() {
  struct point p[3];
  for (int i = 0; i < 3; i++) {
    p[i].x = i;
    p[i].y = 2*i;
  }

  FILE* fp = fopen("points.bin", "w");
  fprintf(“%.2f %.2f, %.2f\n”, p[0][0], p[0][1], p[0][2]);
  fprintf(“%.2f %.2f, %.2f\n”, p[1][0], p[1][1], p[1][2]);
  fprintf(“%.2f %.2f, %.2f\n”, p[2][0], p[2][1], p[1][2]);
  fclose(fp);
}
```

# Demo: Reading binary files

```
struct point {
  int x;
  int y;
};

int main() {
  struct point p[3];

  FILE* fp = fopen("points.bin", "rb");
  fread(p, sizeof(struct point), 3, fp);
  fclose(fp);

  for (int i = 0; i < 3; i++) {
    printf("p[%d] = (%d,%d)\n", i, p[i].x, p[i].y);
  }
}
```