15-213 Recitation

22 January, 2001

Urs Hengartner

Organization

- Email: uhengart+@cs.cmu.edu
- Phone: x8-7571
- Office hours: Tuesday 10:30-11:30am, Wean Hall 4103
- Section A: 10:30-11:20am
  Section B: 11:30–12:20am (together with Umut)
- Send me suggestions for recitation, complaints,…

Overview

- Programming in C
  - Pointers / typecasting / pointer arithmetic
  - sizeof() operator
  - printf() function
- Bit manipulations
- Log in to FISH machines

Pointers

- A pointer variable is a variable whose value is the **address** of a memory location where a specific data type is stored.

```
int *a; // integer-pointer variable
char *r; // character-pointer variable
```

- Before it is used, pointer variable must be assigned address of array, of some other variable, or of dynamically allocated space.

```
int m[10], n;
int *a = m;
int *b = &n;
int *c = (int*) malloc(k * sizeof(int));
```

& and * Operators

- Address-of operator &

```
int m[10], n;
int *a = &m[3]; // a points to address of int
                        array entry m[3]
int *b = &n;    // b points to address of n
```

- Value-of operator *
  `*a = 5` is the same as `m[3] = 5`
  `n = 4 * *b` is the same as `n = 4 * n`
- **Note:** * does not change value of pointer variable, e.g., a still points to `m[3]` after assignment

Dynamic Memory Allocation

- Use **malloc()** and **sizeof()** for dynamically allocating memory

```
int *a = (int*) malloc(k * sizeof(int))
```

Don't:

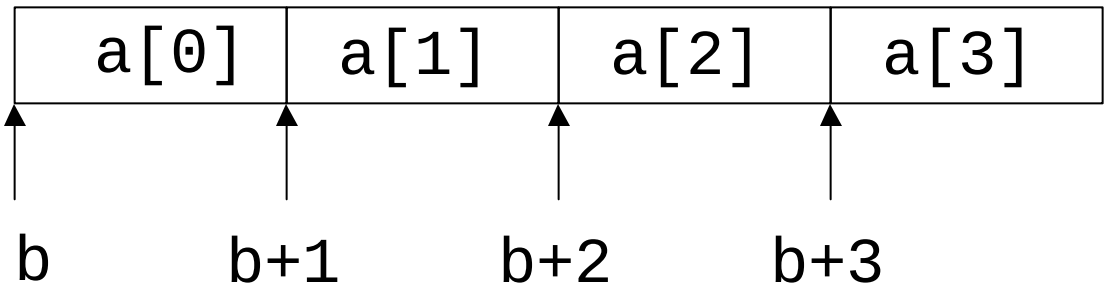
```
int *b = (int*) malloc(k * 4)
                // not portable
```

- malloc() returns generic pointer (`void *`), thus **typecast** it to (`int *`)
- Free dynamically allocated memory with **free()**

```
free(a);
```

Pointer Arithmetic I

- Use pointer arithmetic to access sequence of data cells
- `int a[4], *b = a;`
  `b+i` points to `(i+1)`th element of `a`



`*(b+n)` is the same as `a[n]`

Pointer Arithmetic II

- Note: `b+3` does not mean adding 3 to `b`, rather adding 3 times **the size of the data cell** to `b`
- Subtraction: inverse to adding
  `int *b = &a[3];`
  `b-1` points to `a[2]`
- Walking through array

```
int sumArray(int *a, int size)
{  int i = 0, sum = 0;
   while (i<size) { sum += *a++; i++;}
   return sum;
}
```

printf()

- C++: implicit formatting

```
cout << “Number “ << i
        << “ String “ << s << “\n”;
```

- C: explicit formatting

```
printf(“Number %d String %s\n”,
          i, s);
```

  `%x`: hexadecimal, `%f`: double

  `%5.2f`: precision (`man printf`)

Example

```
void show_bytes(unsigned char *s, int m)
{
    int i;
    for (i=0; i<m; i++)
         printf(“ %.2x”, s[i]);
    printf(“\n”);
}

void show_int(int x)
{
    show_bytes((unsigned char *) &x,
                       sizeof(int));
}
```

Bit Manipulations I

- Logical Operators
  - &&, ||, !
  - Zero means false, non-zero means true
- Bit Operators
  - &, |, ~, ^, <<, >>
  - Arithmetic vs. logical shift
- Masks:
  - lowest byte: 0xff
  - highest byte (32bit): 0xff << 24
  - all 1: ~0

Bit Manipulations II

- Go through examples in handout
- **isEqual(x,y)** with only !, |, &, and ~
  - notEqual for 0/1, 1/0, 0/0:
    x | y
  - correction for 1/1:
    ~ (x & y)
  - together:
    (x|y) & ~ (x & y)
  - isEqual(x,y):
    !((x|y) & ~ (x & y))

Bit Manipulations III

- ```
  int reverseBytes(int x)
  {
      int result = (x & 0xff) << 24;

      result |= ((x >> 8) & 0xff) << 16;
      result |= ((x >> 16) & 0xff) << 8;
      result |= ((x >> 24) & 0xff);

      return result;
  }
  ```
  ```

FISH machines

- Instructions available on course web page
- Before logging in for first time, run **checkin script**:
  `/afs/cs/academic/class/15213-s01/bin/checkin`
- Login: `ssh -l bovik@andrew.cmu.edu FISH.cmcl.cs.cmu.edu`
  `bovik@andrew.cmu.edu@FISH's password: [andrew password]`
- Do not touch files in directory 15-213
- Work in directory 213hw, else keep your files secret:
  `fs sa -clear -acl bovik rlidwka –dir <dirname>`