CPS 104 Recitation: Computer Organization

About Me:
Marisabel Guevara
http://www.cs.duke.edu/~mg/

Slides can be downloaded from the course website

If you have questions:
1. Post them on the Blackboard
2. Come to my office hour
3. Other questions: email me mg@cs.duke.edu

About the recitation
CPS 104 Recitation: Computer Organization

C Programming

Professor: Alvin Lebeck
TA: Marisabel Guevara

mg@cs.duke.edu
Learning Resources

Recommended Book:
The C Programming Language (2nd Edition) by Brian W. Kernighan, Dennis M. Ritchie

The GNU C Library:

Learning C from Java:
http://www.comp.lancs.ac.uk/~ss/java2c/

Other on-line references
http://computer.howstuffworks.com/c.htm

........
Outline

- Part 1: Basic Stuff

- Part 2: Pointers
  - Pointers and Arrays
  - Pointers and Strings

- Part 3: Structures
Outline

Part 1: Basic Stuff
Difference between C and Java

- **Execution Model:**
  - C compiled to machine code on target machine
  - Java compiled to Java Virtual Machine byte code

- **Memory management:**
  - Java – objects with garbage collection
  - C – all in DIY, need to manually allocate memory space

- **Low level machine access:**
  - Java – does not support explicit access
  - C – pointers

- **S/W engineering:**
  - Java – OO
  - C – no objects or OO
Why C?

- More efficient than Java
  - no bounds checking on arrays, arrays initialize to garbage, no object overhead, better code, no emulation/JIT compilation needed, no garbage collection (GC)
- Low Level machine access – OS needs to take into account hardware details
- Full control over memory management – no objects/GC in the way! Address of data in memory never changes (Java GC can change object location)
The First C Program

#include <stdio.h>

void main()
{
    printf("\nHello World\n");
}

On linux machines:
1. Save the code in the file hello.c, then compile it by typing:

    gcc hello.c
    or
    gcc -o hello.out hello.c

2. This creates an executable file a.out, run it.

    a.out
    or
    hello.out

On windows machines:
    Eclipse, Turbo C, Visual Studio
Types, Operators, and Expressions

Variable Names:

- Names are made up of letters and digits; the first character must be a letter. The underscore “_” counts as a letter.

- Upper case and lower case letters are distinct.

- Key words are reserved.

eg:
student_name ✓
#33 x
Types, Operators, and Expressions

Data Types and Sizes:
- char
- int
- float
- double
- short/long
- signed/unsigned

For example:
- [signed] int, 16 bits, -32768~32767
- Unsigned long [int], 32 bits, 0~4294967295
Constants

eg:

#define MAXLINE 1000
char line[MAXLINE+1]

#define VTAB '\007' /*ASCII bell character*/

\n new line \010'
printf("\nHello World\n");

\t horizontal tab \009'
Declarations

Declare the variable before you use it:

eg1:
```
int student;
student = 30;
```
or
```
int student = 30;
```

eg2:
```
int a, b, c;
c = 5;
```
or
```
int a, b, c=5;
```
Global variable vs Local variable

- Global: declared outside of functions, visible in all functions in this file and possibly other files (through extern declaration)
  lifetime -- as long as entire program
- Local: declared within a function, visible only inside defining function
  lifetime – as long as invocation of function only
Example

```c
int sum;

f() {
  int sum;
  sum++;
  ...
}

g() { ... sum += 7; ... }
```
Operators

Arithmetic Operators:
  +  -  *  /  %

eg1:
  x=7%4; result: x=3

Relational Operators:
  >  <  >=  <=  ==  !=

eg2:
  (c=getchar()) != '\n'

Logical Operators:
  !  &&  ||

eg3:
  if (!valid)
  if (valid == 0)
Type Conversions

(double)a
(int)(x+y)
(int)x+y

eg:

```c
main()
{
  float x;
  int i;
  x=3.6;
  i=(int)x;
  printf("x=%f, i=%d", x, i)
}
```

result:

x=3.600000, i=3
Increment and Decrement Operators

++i; --i; /*add/deduct before use */
i++; i--; /* use before add/deduct */

eg1:
    i = 3;
    printf("%d", ++i);
result: i=4

eg2:
    i=3;
    printf("%d", i++);
result: i=3
Bitwise Operators:

& | ^  <<  >>  ~

eg:

3 & 5 =?

00000011

& 00000101

result 00000001
Expressions

Assignment Operators and Expressions:

eg:

\[ x *= y + 1 \]
\[ x=x*(y+1) \]

Conditional Expressions:

expr1 ? expr2 : expre3

eg:

\[ z = (a>b) ? a : b; /* z = \text{max}(a, b) */ \]
#include<stdio.h>

Formatted I/O:
printf(“%d\n”, i);
scanf(“%d”, &a);
Control Flow

If
  eg:  
    if(x>y)
    printf("%d", x);

If-Else
  eg:  
    if(x>y)
    printf("%d", x);
    else
    printf("%d", y);

If-else-if-else .....if else
Control Flow

Switch

eg:

switch(grades)
{
  case 'A': printf("85~100\n"); break;
  case 'B': printf("70~84\n"); break;
  case 'C': printf("60~69\n"); break;
  case 'D': printf("<60\n"); break;
  default: printf("error\n");
}
Control Flow

While

eg1:
int i=1, sum=0;
while(i<=100)
{
    sum=sum+i;
    i++;
}

Do-while
eg2:
int i=1, sum=0;
do
{
    sum= sum+i;
    i++;
} while(i<=100)

What if we set i=101 at the beginning?
Control Flow

For

eg1:

```
for(i=1;i<=100;i++)
    sum=sum+i;
```

eg2:

```
for(i=1; ; i++)
    sum=sum+i;
```

eg3:

```
for(int i=1;i<=100;i++) x
```

Not allowed in C
Control Flow

break;

It provides an early exit from for, while and do, just as from switch. It causes the innermost enclosing loop or switch to be exited immediately.

eg:
for(r=1; r<=10; r++)
{
    area=pi*r*r;
    if(area>100)
    {
        break;
        printf("%f", area);
    }
}
continue;

It causes the next iteration of the enclosing for, while, or do loop to begin. It applies only to loops, not to switch.

eg:

```c
for(n=100; n<=200; n++)
{
    if(n%3==0)
        continue;
    printf("%d", n);
}
```
Control Flow

goto and Labels:

loop: if(i<=100)
    {
        sum=sum+i;
        i++;
        goto loop;
        printf(“%d”, sum);
    }
Functions and Program Structure

Eg:
max(int x, int y)
{
    int z;
    z=x>y? x:y;
    return(z);
}
main()
{
    int a,b,c;
    scanf("%d,%d", &a, &b);
    c=max(a,b);
    printf("Max is %d", c);
}
Pass by value VS Pass by reference

- What is the difference?
- Java: Pass by reference, except for primitive data type (eg. int, etc)
- C: Pass by value
  
  Can also pass by reference by using ‘&’ operator
  eg.
  ```c
  int modify(int *x) { *x = x+1; }
  modify(&data);
  ```
int mystery1(int n) {  
  int bit;  
  if (n == 0) return 0;  
  else {  
    if (n & 1) bit = 1;  
    else bit = 0;  
    return bit + 2*mystery1(n/2);  
  }  
}  
What is the output for mystery1(42)?
Outline

Part 2: Pointers
Pointers and Arrays

Declaration of an array:

eg1:
  int a[10];

eg2:
  int n;
  scanf("%d", &n);
  int a[n];
Pointers and Arrays

One-Dimensional Array:

eg1:

```c
int i, a[10];
for (i=0; i<10; i++)
a[i]=5;
```

eg2:

```c
int i, a[10];
for (i=0; i<10; i++)
    scanf("%d", &a[i]);
```

eg3:

```c
int a[10]={2,4,5,6,2,4,8,9,1,2};
```
Pointers and Arrays

Two-Dimensional Array:

eg1:

```c
float a[3][4];
```

eg2:

```c
int a[3][4]= {{1,3,4,6},{5,6,7,4},{3,6,8,9}};
```
Pointers and Arrays

Pointers:
A pointer is a memory location that contains the address of another memory location.

Two operators:
& : “address of” (NOT bitwise AND operator)
* : indirection / dereferencing operations
Pointers and Arrays

eg1: (from Professor Lebeck's slides)

```c
int x;
int *p; /* p is a pointer pointing to an integer*/
p=&x;
x=2;
Then:
  *p=2;
p=?
```

eg2:
```
int a; int *p;
a=100;
p=&a;
printf("%d\n", a);
printf("%d\n", *p);
result:? 
```
Pointers and Arrays

- From previous example:
  eg1:
  ```c
  int x=2;
  int *p; /* p is a pointer pointing to an integer*/
  p=&x;
  *p=3;
  Then:
  x=?;
  ```

- Now, we use pointers: Pointers pointing to the elements in an array:
  eg2:
  ```c
  int a[10];
  int *p;
  p=&a[0]; /* p points to the first element in the array. */
  or
  p=a; /* In C, the name of the array represents the address of the first element in the array*/
  ```

  eg3:
  ```c
  int *p = &a[0];
  or
  int *p=a;
  ```
Pointers and Arrays

What is p+1? a+1?

What is *(p+1) ? *(a+1) ?

eg:

```c
main()
{
    int a[10];
    int *p, i;
    for(i=0; i<10; i++)
        scanf("%d", &a[i]);
    printf("\n");
    for(p=a; p<(a+10); p++)
        printf("%d", *p);
}
```

result: 1 4 6 7 8 6 4 4 6 9
1 4 6 7 8 6 4 4 6 9

Pointers to Multi-dimensional arrays (after class)
Pointers and Strings

eg1:
    char c[3] = {'a', 'b', 'c'};
or
    char c[3];
    c[0] = 'a'; c[1] = 'b'; c[2] = 'c';

eg2:
    char str[] = {"I am happy"};
or
    char str[] = "I am happy";
or
    char str[] = {'I', ' ', 'a', 'm', '', 'h', 'a', 'p', 'p', 'y', '\0'};

eg3:
    char str[] = {"USA"};
    printf("%s", str);
    In the memory: USA\0
    Do NOT print '\0'
Pointers and Strings

String Functions in C:

puts(str);
gets(str);
strcat(str1, str2);
strcpy(str1, str2);
strcmp(str1, str2);
strlen(str);
strlwr(str);
strupr(str);
strupr(str);

eg:
    char str[15]="I am happy":[
    printf("%d", strlen(str));
result: 11
    NOT 15, NOT 10
Pointers and Strings

- Recall previous example:
  
  eg1:
  
  ```c
  char str[]="I am happy";
  printf("%s", str);
  ```

- Now, we use pointers: Pointers pointing to a string:
  
  ```c
  char *p="I am happy";
  printf("%s", p);
  ```
  
  or
  
  ```c
  char *p;
  p="I am happy";
  ```
Pointers and Strings

eg:

```c
main()
{
    char a[]="I am happy", b[20], *p1, *p2;
    int i;
    p1=a; p2=b;
    for(; *p1!='\0'; p1++, p2++)
        *p2=*p1;
    *p2='\0';
}
```
Outline

Part 3: Structures
Structures

eg:

```c
struct student
{
    int id;
    char name[20];
    float score;
}stu={1009, “Albert”, 98.0};
```

1009 Albert 98.0
Structures
Arrays of Structures:

```c
struct student
{
    int id;
    char name[20];
    float score;
};
struct student stu[3]=
{{100, “Jen”, 87},
{101, “Steve”, 98},
{102, “Bob”, 95}};

or

struct student
{
    int id;
    char name[20];
    float score;
}stu[3]=
{{100, “Jen”, 87},
{101, “Steve”, 98},
{102, “Bob”, 95}};
```
Arrays of Structures:

eg1:

    printf("%s\n", stu[2].id);

or

    printf("%d\n", stu[2]->id);

eg2:

    scanf("%d",&stu[2]->id);
Structures

Pointers and Structures:

eg:
```c
main()
{
    struct student{
        int id;
        char name[20];
        float score;
    };
    struct student stu;
    struct student *p;
    p=&stu;
    stu.id=208;
    printf("%d\n", stu.id);
    printf("%d\n", (*p).id);
}
```
Structures

Pointers and Structures:

eg:

```c
struct student{
    int id;
    char name[20];
    float score;
    struct student *next;
};
```

next is a pointer pointing to a struct student.
Structures

Array of Classes (Linked List)

eg:
struct node{
    int me;
    struct node *next;
};
struct node ar[10];     /* In C++, node *ar= new node[10];*/
main(){
    int k;
    struct node *p;
    p=ar;
    for(k=0; k<9; k++)
    {
        p->me=k;
        p->next=&ar[k+1];
        p++;
    }
    p->me=9;
    p->next=NULL;
    p=&ar[0];
    while(p!=NULL)
    {
        printf(“%d %x %x\n”, p->me, p, p->next);
        p=p->next;
    }
}
Homework Related

HW1, PART 2, Binary Tree(Memory)

• Binary Tree: a tree data structure in which each node has at most two children. Typically the child nodes are called left and right.
  struct node{
    struct node *left;
    struct node *right;
  };

• A binary tree of depth 4 (i.e., 8 leaf nodes) – full tree
  struct node ar[15];

• Pre-order (i.e., parent, left-child, right-child), use recursion traverse
Homework Related

HW1, PART 2, Bit Manipulation

• A 32-bit unsigned number: Unsigned long [int], 32bit, 0~FFFFFFFF

• Pad it with 0's: unsigned long input, output;

• Input: scanf("%X", &input); printf("\n%X\n", output);

• Bit shift operation