C Basics, Arrays, Functions & IO

15-123
Systems Skills in C and Unix

Today's Learning Objectives

- **We will do a brief intro to**
  - **basic Data types and Formats**
  - **How Conditionals and loops work**
  - **How Arrays are defined, accessed, and stored**
  - **How Functions work**
  - **How Strings are defined, accessed, changed**
  - **How to perform File I/O**
- **Disclaimer:** All of these topics need deeper investigation. We will do that as the course progresses

Announcements

- Finding unix/C help
  - Unix man pages is your friend
    - > man ls
    - > man stdio.h
- Fill out the background survey and complete "prior knowledge" salon (if you have not done so)
- Start working on Lab1 Salon (graded)
  - Due Friday 1/21/11
- Think about Lab 1
  - Due Sunday 1/23/11 (graded)
- SL1 is due today 1/18/11 Tuesday (graded)
  - Ascii table on course web site

From last lecture

- Representing integers in base b
- Modular arithmetic
  - x = y mod (n) ↔ n / (x-y)
- 2' s compliment representation of negative numbers
- Ranges
  - max, min for types int, short and long
  - max for unsigned int, short and long

Representing integers in base b

- Efficient calculation of a sum given the digits and the base

Modular Arithmetic

- integers and ints
  - Not the same
  - ints have fixed precision
  - Machines have word length
  - ints perform all operations +, -, / , * in 32-bit (or 64-bit)
- Definition of x = y mod (n)
- What does it mean to say x + y = 0 mod (n)?

Modular Table (4-bits)

- 16 = 0 mod(16) ←→ -16 = 0 mod (16)
- 17 = 1 mod(16) ← → -15 = 1 mod (16)
- 18 = 2 mod (16)

- Also we can write
  - -1 = 15 mod (16)
  - Hence in modular [16] arithmetic -1 = 15

2's compliment representation of negative numbers

- Definition of
  - One's compliment
  - Two's compliment
  - Binary additions
- Computers can perform subtraction using 2's compliment

max/min of unsigned/signed ints

- signed 32-bits ints
  - max
  - min
- unsigned 32-bit ints
  - max
  - min

Brief intro

Conditionals & Loops

```
if (condition) {   }
if (condition) { } else { }
if (condition) { } else if (condition) {  } else { }
switch  (condition) {
  case 1 :  ... ; break;
  case 2 :  ....; break;
   .....
  default: ....;
}
```

- for (initial condition; exit condition; loop control) { loop body }
- while (condition) { loop body }

Loop semantics

- for (int i=0; i<n; i++) { loop body }
- i=0; while (i<n) { ....; i++; }
- Assuring that loop ends

Writing output to STDOUT/File

- Prototype
  - int printf (const char * format, ...);
  - int fprintf (FILE*, const char * format, ...);
- ***Example***
  - ***printf("This is a test %d %.4f %10.2f %c\n",134,56.455, 3355.5346, 65);***
  - ***File output***
    - ***FILE* fp = fopen("filename", "w");***
    - ***fprintf(fp, "This is a test %d %.4f %10.2f %c\n",134,56.455, 3355.5346, 65);***

Print Formats

specifier	Output	Example
c	Character	a
d or i	Signed decimal integer	392
e	Scientific notation (mantise/exponent) using e character	3.9265e+2
E	Scientific notation (mantise/exponent) using E character	3.9265E+2
f	Decimal floating point	392.65
g	Use the shorter of %e or %f	392.65
G	Use the shorter of %E or %f	392.65
o	Signed octal	610
s	String of characters	sample
u	Unsigned decimal integer	7235
x	Unsigned hexadecimal integer	7fa
X	Unsigned hexadecimal integer (capital letters)	7FA
p	Pointer address	B800:0000
n	Nothing printed. The argument must be a pointer to a signed int, where the number of characters written so far is stored.	
%	A % followed by another % character will write % to stdout.	

Source: cplusplus.com

Reading Data

- **scanf("%d", &x);**
- int **getchar(void)**
  - getchar returns the ASCII value of the next input character or EOF, a constant defined in stdio.h
- sscanf. The prototype for sscanf is:
  - **int sscanf(char*S,char* format, arg1, arg2,...)**

Reading from a file

```
FILE*  fp = fopen("filename", "r");
int x;
while (fscanf(fp,"%d", &x)>0) {
    ......
}
```

- See man fscanf (on how to use fscanf)
  - What are the arguments?
  - What does it return?

Writing to a file

```
FILE*  fp = fopen("filename", "w");
/* writing from an array to a file */
for (int i=0; i<n; i++)
  fprintf(fp, "%d", A[i]);
```

- See man fprintf
  - What are the arguments?
  - What does it return?

Arrays in C

- Declarations
  - <type> arrayName[size];
  - int A[ ] = {1,2,3}; /* implicit size definition*/
  - char A[10];
- Name of the array is the address of the first element of the array (a const pointer)
- Access
  - A[i] = 10;
  - printf("%d", A[i]);
  - Is it possible to say: A = {1,2,3}; ?
- Arrays are statically allocated
  - Fixed size
- Resizing arrays will be discussed later

2D Arrays

- <type> nameOfArray[rows][cols];
- Arrays are allocated as a block of m*n*sizeof(type) bytes. Consider: int A[5][2];

Strings in C

- A valid C String is an array of char's ending with NULL character '\0'

G	U	N	A	'\0'

- C strings are mutable

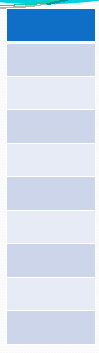
Functions

- <return_type> function_name(type1 n1, type2 n2, ...);
- Suppose swap is a function defined as follows

```
void swap(int x, int y){
   int tmp = x;  x = y;  y = tmp;
}

int x=10, y=15;
swap(x,y);
/* what are the values of x and y */
```

How functions work

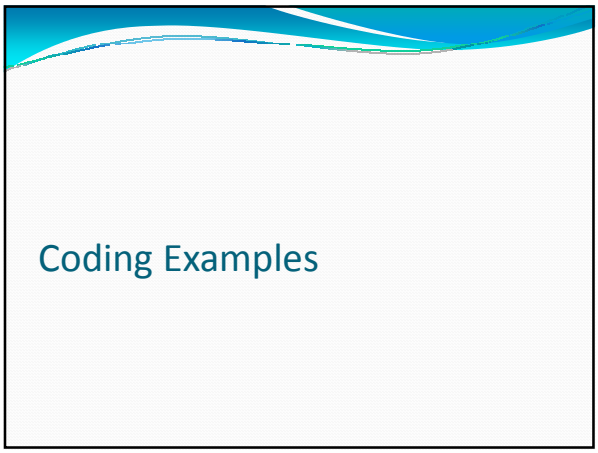
```
int main(int argc, char* argv[]){
    int x=10, y=15;
     swap(x,y,);
    /* next instruction */
}

void swap(int x, int y){
   int tmp = x;  x = y;  y = tmp;
}
```



Runtime stack

Not all functions are created equal

- Functions have overhead
  - Runtime stack use
- Some functions are straight forward and some are not
  - Inductive definition of power(x,y)
  - Fast algorithms for computing power(x,y)
- Function efficiency "matters" in this course


Coding Examples