OS Lab Tutorial 1

Linux System and Basic C Programming

Use Linux System

- Connect to the Linux machine on campus from your own laptop
  - Mac,Linux users could use OpenSSH and type the following commands in the terminal:
    - ssh netlinkid@hostname
  - Host name could be "linux.csc.uvic.ca" or the machine's name in the ECS242(e.g., u-fedora.csc.uvic.ca)
  - Windows users could install "putty" and configure it to be connected
    - "putty.exe" uses a terminal for interaction

Basic File System Operations

- Basic commands
  - **man**
  - **ls**: list directory contents
  - **pwd**: print working directory
  - **cd**: change directory
  - **cp**: copy files from source to dest
  - **mv**: cut and move files from source to dest
  - **mkdir**: create a directory
  - **rmdir**: remove a directory
  - **rm**: remove files
  - **chmod**: change file mode bits
  - **exit**
- Try it yourself !
  - For details about options, type in the terminal:
    - man 1 ls
    - man 1 pwd
    - ...

Unix Manual ($ man [section] [name]**)**

- Section 1: user commands
  - $ man 1 ls
  - $ man 1 printf
- Section 2: system calls
  - $ man 2 fork
  - $ man 2 getcwd
- Section 3: general-purpose functions to programmers
  - $ man 3 printf
- For a complete description of man page, just type:
  - $ man man
- It will be very helpful throughout this course. Don't forget it !

Why should we learn C?

- Better control of low-level operations
- Better performance
- Other languages, like Java and Python, hide many details needed for writing OS code
  - Memory management
  - Error detection
  - ...

C Programming under Linux - Editor

- Vim
  - It works in command line mode
  - For a quick tutorial(nearly 40 mins), you could type the following command and try it yourself.
    - $ vimtutor
- Gedit
  - It has GUI
  - $ gedit sample.c
- Others: Emacs,...

Compile your C programs - GCC

- Basic Usage
  - $ gcc test.c -o test
  - $ ./test
- gcc working process (test.c)
  - preprocessing
    - gcc test.c -o test.i -E
  - compilation
    - gcc testi -o test.s -S
  - assembly
    - gcc test.s -o test.o -c
  - linking
    - gcc test.o -o test

```
// test.c
#include <stdio.h>
int main()
{
        printf ("Hello, OperatingSystem.\n");
        return 0;
}
```

Compile your C programs - GCC

- -Wall option
  - $ gcc -Wall test.c -o test
  - We suggest that you always add this option when you compile your program. This option enables all compiler's warning information. It helps you write better code.
- Click following link for a complete documentation of GCC
  http://gcc.gnu.org/onlinedocs/

Debug your C programs - GDB

- $ gcc -g test.c -o test: option -g adds debugging information when creating the executable file
- Commands:
  - $ gcc -g test.c -o test
  - $ gdb test
  - (gdb) list
  - (gdb) run
  - (gdb) break
  - (gdb) next
  - (gdb) step
  - (gdb) clear
  - (gdb) watch
  - (gdb) info watch/break
  - (gdb) help
- Official documentation
  - http://www.gnu.org/software/gdb/documentation/

C Programming under Linux - Makefile

- Two .c files: main.c add.c
- main.c

```c
#include<stdio.h>
#include "add.h"

int main()
{
   int a=2,b=3;
   printf("the sum of a+b is %d\n", add(a,b));
   return 0;
}
```

- add.c

```c
int add(int i, int j)
{
   return i + j;
}
```

add.h

```c
int add(int i, int j);
```

Makefile Example

- How to get an executable file from two source files ?
  - $ gcc -c main.c -o main.o
  - $ gcc -c add.c -o add.o
- Be careful, it won't work if you use either of
  - gcc main.c
  - gcc add.c
- Finally,
  - $ gcc main.o add.o -o test

- We can write a *Makefile* to handle each of the steps
- Then, use make to compile all the files

Makefile Example

- Basic Syntax
  - target: denpendencies

    commands
  - official document http://www.gnu.org/software/make/manual/make.html#Introduction
- Sample (Create a file named "Makefile")

```
test: main.o add.o
	gcc main.o add.o -o test
main.o: main.c add.h
	gcc -c main.c -o main.o
add.o: add.c add.h
	gcc -c add.c -o add.o
```

- $ make

More on C Programming Language

- Simple Data Type

Name	# of Bytes (typical)	range	format
int	4		%d
char	1		%c
float	4		%f
double	8		%lf
long	4		%l
short	2		%i

- You don't need to remember the range. You can simply print them!

Print the scope of a data type

```
#include <stdio.h>
#include <limits.h>

int main()
{
    printf("Minimum Value of Signed Int(type) : %d\n", INT_MIN );
    printf("Maximum Value of Signed Int(type): %d\n", INT_MAX );
    return 0;
}
```

- Check "limits.h" for more.

Secondary Data Type

- Array
  - int a[5] = {1,2,3,4,5};
  - char b[5] = {'a','b','c','d','e'};
  - char c[] = "abcd"
    - In C, strings are terminated by '\0'
    - So the array c will have 5 elements (c[0]~c[4])
    - c = 'a' 'b' 'c' 'd' '\0'

The difference between single quote and double quote

- Single quote is used for single character
  - char a = 'a';
- Double quote is used for string
  - char s[5] = "abcde";

Secondary Data Type

- Pointers
  - int a = 3;
    - A 4-byte memory space will be allocated for the variable "a". Pointer can be used to store the address of such block of memory space
  - int *p = &a;
    - int * means p is a pointer which points to an integer. "&" is used to get the address of the variable "a". Now, p stores the address of a.
    - You can use gdb to print the address (print p)
  - printf("The value of a is: %d", *p);
    - You can access the value of "a" by *p. Without such a star(*), p is the memory address of "a".

Secondary Data Type

- Pointers
  - char a[] = "abcd";
    - Specifically, "a"(without the subscript index) stores the beginning address of the char array
    - That means you can print the array using
      - printf("The array is: %s", a);
    - The name of an array is a constant while the pointers are variables.
      - a ++; // wrong
      - pointer ++; // correct
  - char *p = a;
    - Now, pointer p points to the array a. p stores the start memory address of a.
    - p[0] is 'a'; p[1] is 'b'

More on Pointers

- Dyanmic Memory Allocation
  - int *aPtr;
    - The address aPtr points to is undefined.
    - *aPtr = 5; will raise a segmentation fault.
  - aPtr = (int *) malloc (sizeof(int));
    - Allocate enough space for an integer. malloc() will return the beginning address of such space to aPtr.
  - *aPtr = 5;
    - Now you can assign an integer to the address
  - free (aPtr);
    - You should free the allocated space before the program stops !
- Use "man malloc" for more information
  - E.g., realloc() changes the size of memory block pointed by a pointer.

Secondary Data Type

- Structure

```
#include <stdio.h>

struct date{
    int month;
    int day;
    int year;
};  // Don't forget the semi-colon here.

int main()
{
    struct date myDate;
    myDate.month = 5; myDate.day = 19; myDate.year = 2012;
    printf("Today's date is %d-%d-%d.\n", \
        myDate.month,myDate.day, myDate.year);
    return 0;
}
```

Secondary Data Type

More on Structure

- Suppose we have defined the *struct date*
- We could then create an array of such type
  - struct date dateCollection[50];
- To access each of element in the array, simply use the index
  - dateCollection[0].month = 5;
  - dateCollection[3].year = 2012;

Using typedef

- typedef int Value;
  - Value a = 5; // The same as "int a = 5;"
- typedef int* ValuePtr;
  - ValuePtr b = &a; // The same as "int *b = &a;"
- typedef struct date Date;
  - Date myDate; // The same as "struct date myDate;"
- typedef struct date * DatePtr;
  - DatePtr myDatePtr; // The same as "struct date * myDatePtr;"

Call by Value VS. Call by Reference

```
void swap1(int a, int b)
{
    int temp;
    temp = a;
    a = b;
    b = temp;
}
```

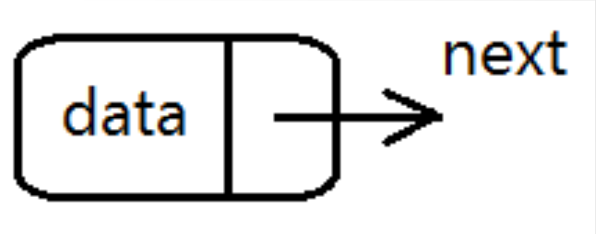
```
void swap2(int *a, int *b)
{
    int *temp = (int *)malloc(sizeof(int));
    *temp = *a;
    *a = *b;
    *b = *temp;
    free(temp);
}
```

```
int main()
{
    int a = 1, b = 2;
    swap1(a,b);
    printf("Call by Value: a = %d, b = %d\n",a,b);
    swap2(&a,&b);
    printf("Call by Reference: a = %d, b = %d\n",a,b);
    return 0;
}
```

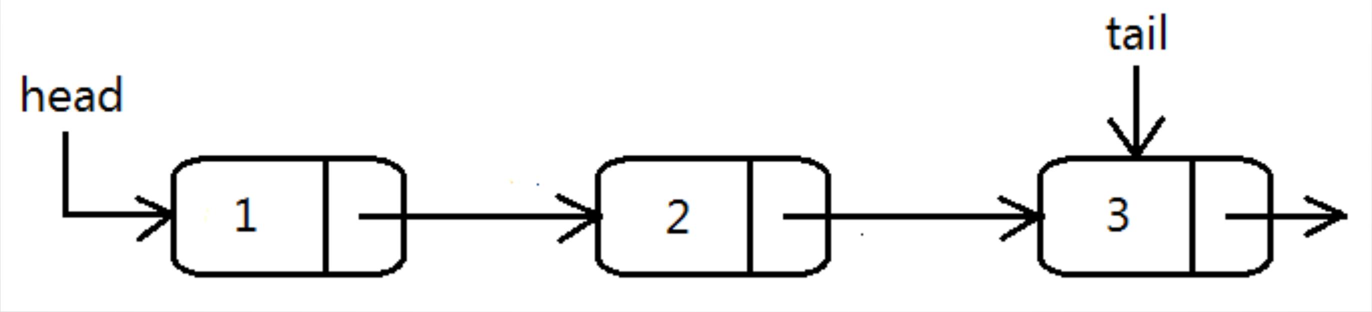
Data Structure - Linked List

- Struct definition

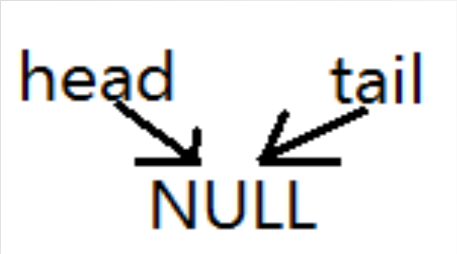
```
typedef struct node
{
    int data;
    struct node *next;
}Node, *NodePtr;
```



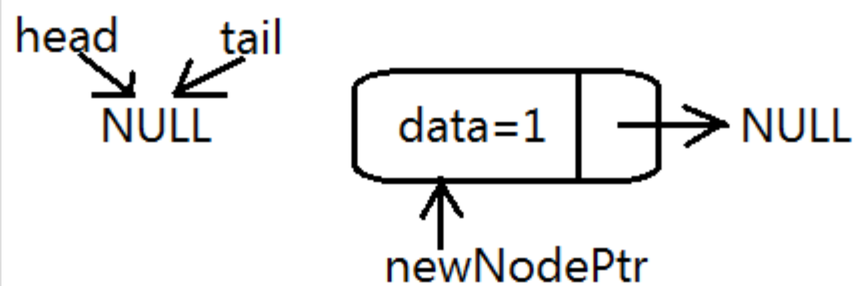
- How do we create a list?



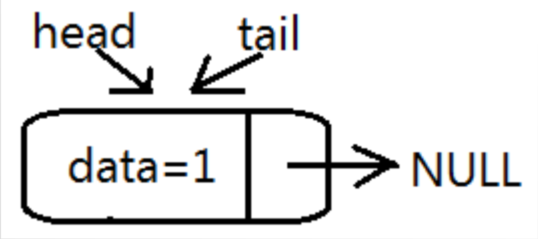
Linked List - Creation and Insertion




```
NodePtr head, tail;
head = tail = NULL; // Initialization
```

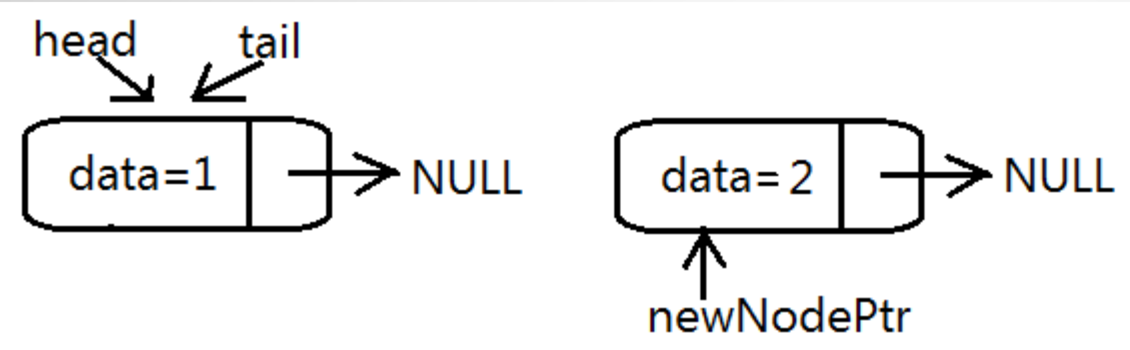



```
NodePtr newNodePtr; // create the first node
newNodePtr = (NodePtr)malloc(sizeof
(Node));
newNodePtr->data= value;
newNodePtr->next = NULL;
```

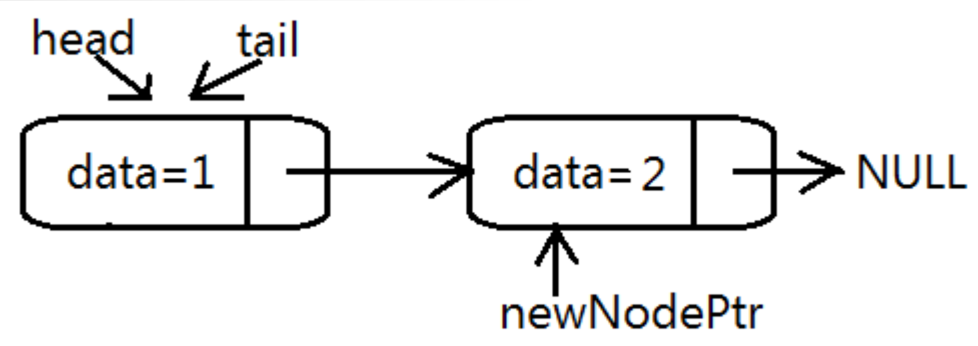



```
if(head == NULL && tail == NULL){
        head = newNodePtr;
        tail = head;
}
```

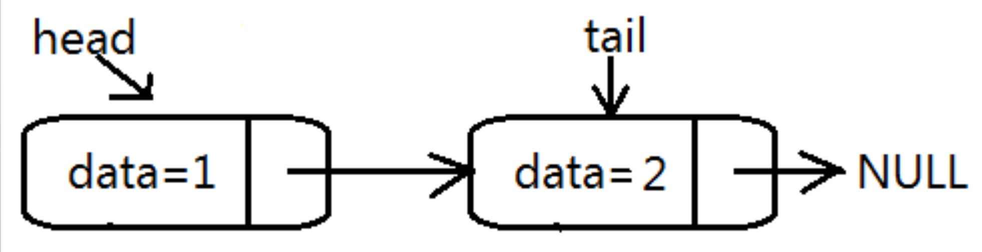
Linked List - Creation and Insertion



// Insert another node

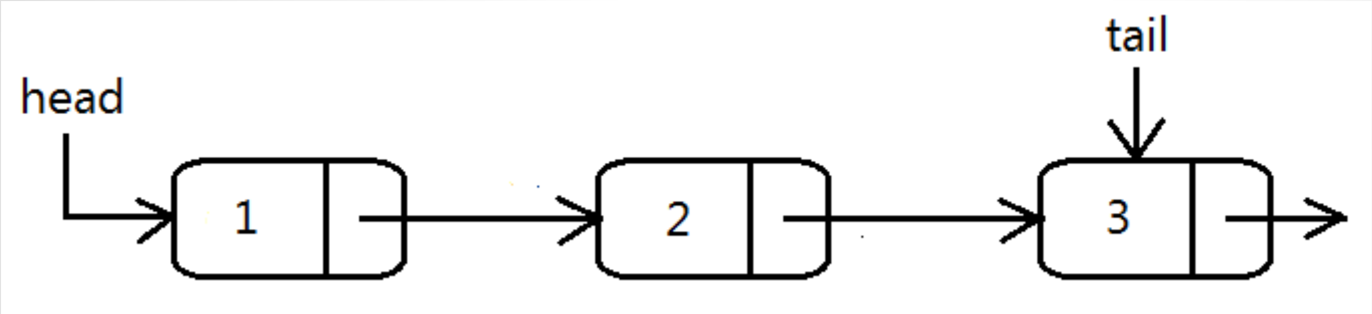


tail->next = newNodePtr;
// Link the new node



tail = newNodePtr;
// tail points to the last node

Linked List - Deletion ?



- For more about linked list, you could refer to a tutorial by Stanford
  - http://cslibrary.stanford.edu/103/

Avoiding Common Errors

- Always initialize anything before you use it (especially the pointers !)
- You should explicitly free the dynamically allocated memory space pointed by pointers
- Do NOT use pointers after you free them
  - You could let them point to NULL
- You should check for any potential errors (It needs much exercise)
  - E.g., check if the pointer == NULL after memory allocation

Sample Code Package

- args.c (read command options)
- sample.c (read one line of input)
- inf.c (background process)

Post Questions in Chat room