Manual Page ($ man [section] [name])
● Section 1: Exes or user commands
  ○ $ man 1 ls
  ○ $ man 1 printf
● Section 2: system calls (kernel)
  ○ $ 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 the assignments. Don't forget it !

$ vimtutor

**Compile your C programs - GCC**
● Basic Usage
  ○ $ gcc test.c -o test
  ○ ($ gcc *.c -o test)
  ○ $ ./test
● gcc working process (test.c) (Reference)
  ○ preprocessing
■ gcc test.c -o test.i -E
  ○ compilation
■ gcc test -o test.s -S
  ○ assembly
■ gcc test.s -o test.o -c
  ○ linking
■ gcc test.o -o test

Compile your C programs - GCC
● -Wall option
  ○ $ gcc -Wall -o test test.c
  ○ We suggest that you always add this option when you compile your program. This option enables all compiler's
warning information. It helps you improve code quality.
● A complete documentation of GCC

**C Programming under Linux - Makefile**
● Two .c files: main.c add.c
● main.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

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

● add.h

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

**Compile without makefile**

● 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

● Then,
  ○ $ gcc main.o add.o -o test

● A trick:
  ○ $ gcc *.c (Generate one executable file from *.c)

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

Makefile Example 1
https://www.gnu.org/software/make/manual/make.html#Introduction

● Basic Syntax
  ○ target: dependencies (or pre-requsite)

[Tab]commands
  ○ official document **http://www.gnu. rg/software/make/manual/make.html#Introduction**

● Sample (Create a file named "Makefile" without extension)

```
	test: main.o add.o
```

● $ make
● Use -f to specify a makefile
○ $ make -f mf-1

```
	gcc -o test main.o add.o
	main.o: main.c add.h
		gcc -c main.c
	add.o: add.c add.h
		gcc -c add.c
	clean:
		rm -f test
		rm -f *.o
```

More on Makefile (reference)

● Automatic Variables

Automatic variables are set by make after a rule is matched. They include:

● $@: the target filename.
● $*: the target filename without the file extension.
● $<: the first prerequisite filename.
● $^: the filenames of all the prerequisites, separated by spaces, discard duplicates.
● $+: similar to $^, but includes duplicates.
● $?: the names of all prerequisites that are newer than the target, separated by spaces.

Makefile Example 2
● Compile multiple sources at one time
○ Sample (Create a file named "Makefile")

```
--------------------
CC = gcc
CFLAGS = -Wall -o

all: test1 test2 test3

test1: test1.c
	$(CC) $(CFLAGS) test1 test1.c
test2: test2.c
	$(CC) $(CFLAGS) test2 test2.c
test3: test3.c
	$(CC) $(CFLAGS) test3 test3.c



---------------------

CC = gcc
CFLAGS = -Wall -o
all: test1 test2 test3
test1: test1.c
	$(CC) $(CFLAGS) $@ $<
test2: test2.c
	$(CC) $(CFLAGS) $@ $<
test3: test3.c
	$(CC) $(CFLAGS) test3 test3.c

------------------
```

● $ make

Debug your C programs - GDB
● $ gcc -g test.c -o test: option -g adds debugging information when creating the executable file
● Commands:

```
$ gcc -Wall -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/

A step-by-step tutorial of basic C programming can be found at:

○ http://web.uvic.ca/~cchenv/ta/csc360/html/linux_basic_c. html

**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'

● 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".

● 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.
- ● 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;
}
```

- ● 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**

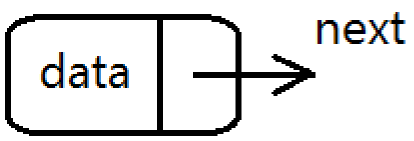
```
void swap1(int a, int b)
{
	int temp;
	temp = a;
	a = b;
	b = temp;
}
void swap2(int *a, int *b)
{
	int *temp = (int *)malloc(sizeof(int));
	*temp = *a;
	*a = *b;
	*b = *temp;
```

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

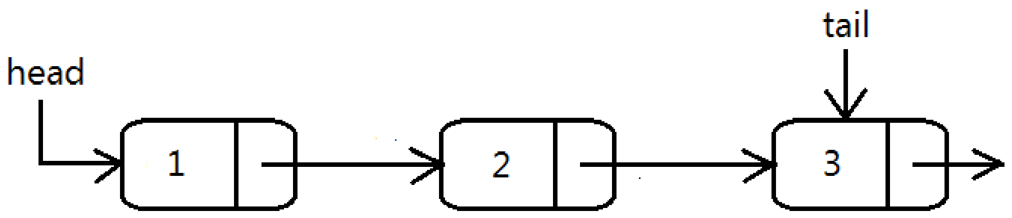
**Data Structure - Linked List**

● Struct definition

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

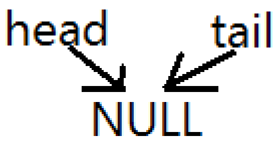


● How do we create a list?

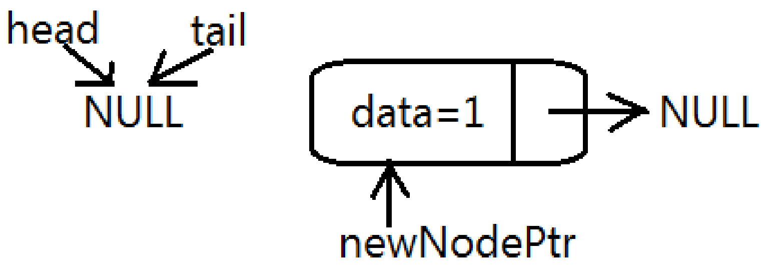


***Linked List - Creation and Insertion***

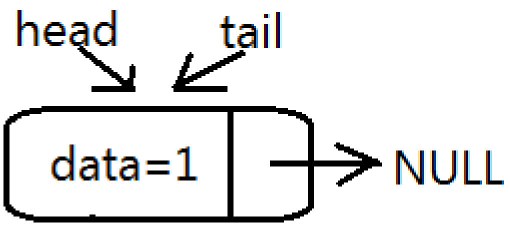
```
NodePtr head, tail;
head = tail = NULL; // Initialization
```

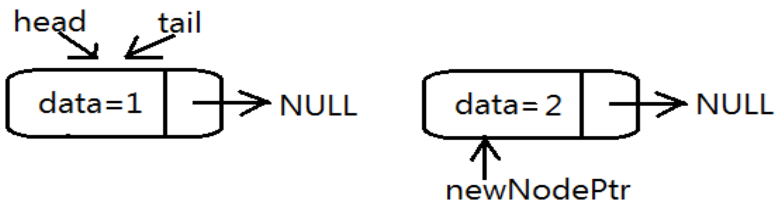


```
NodePtr newNodePtr; // create the first node
newNodePtr = (NodePtr)malloc(sizeof(Node));
newNodePtr->data= value;
newNodePtr->next = NULL;
```



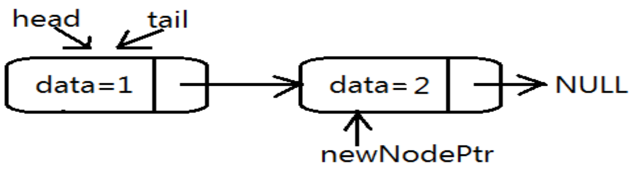
```
if(head == NULL && tail == NULL){
        head = newNodePtr;
        tail = head;
}
```

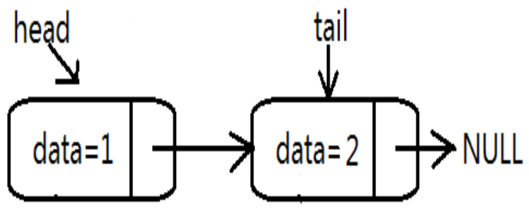




// Insert another node

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





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

**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

**OS Lab Tutorial**

**Thread**

● Pthreads

○ Create/Terminate

○ Obtain the returned value

● Synchronization

○ mutex

○ cond

● Some Programming Tips

Data Source: https://computing.llnl.gov/tutorials/pthreads/ Code: https://computing.llnl.gov/tutorials/pthreads/fork_vs_thread.txtPOSIX Threads (pthreads)

● POSIX - Portable Operating System Interface

○ maintain the compatibility between operating system

○ The standard, POSIX.1c, Threads extensions (IEEEStd 1003.1 c-1995), defines an API for creating and manipulating threads.

● Standard Unix threading library

● Provides useful concurrent constructs

○ mutex

○ conditional variables

● Note

○ The POSIX semaphore API works with POSIX threads but is not part of threads standard. The semaphore procedures are prefixed by "sem_" instead of "pthread_" ( http://en.wikipedia.org/wiki/POSIX_Threads )pthread routines

● pthread_attr_init, pthread_attr_destroy

○ Initialize/destroy the attribute object of a thread

○ joinable/detached

● pthread_create

○ Create a thread

● pthread_join

○ wait for threads to finish

● pthread_exit

○ finish a thread

● pthread_cancel

○ Cancel a threadFound no Man Page Info. ?

● Man Pages

○ The manual documents for POSIX pthread library may not be installed on some versions of the Linux system. You may have to use the online version or install it by yourself:

■ Linux Man Pages: http://linux.die.net/man/

● Search by "Names" = pthread

■ Installation (Ubuntu):

● $ sudo apt-get install manpages-posix-dev

● $ sudo apt-get install glibc-doc

■ Then check the installation by:

● $ man pthread_createpthread_create

● <pthread.h>

● int pthread_create(pthread_t *restrict thread, const pthread_attr_t *restrict attr, void *(*start_routine)(void*), void *restrict arg);

● Arguments

=>pointer to a thread ID, type: pthread_t

=>attribute object of a thread (could be NULL)

=>routine to be executed in a thread

=>arguments for the routine (could be NULL)

```
// Create a detached thread
        pthread_attr_t attr; // thread attribute

// set thread detachstate attribute to DETACHED
        pthread_attr_init(&attr);
        pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_DETACHED);

// create the thread
pthread_create(&tid, &attr, start_routine, arg);pthread_create

// From apue 2
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
void *print_sth( void *ptr )
{
        char* message; pthread_join( thread1, NULL);
        message = (char*)ptr;
        printf("%s\n",message); return 0;
        return NULL;
}

int main()
{
        pthread_t thread1;
        char *message1 = "Thread 1";
        iret1 = pthread_create( &thread1, NULL, print_sth, (void*) message1);
        printf("Thread 1 finished.\n”);
return 0;
}
```

**pthread_join**

● Wait for a (joinable) thread to finish. If return value is not NULL, it copies the value into a location pointed to by *retval

● <pthread.h>

● int pthread_join(pthread_t thread, void **retval);

● ***Arguments***

○ id of the created thread

○ returned value from the created threadpthread_join

```
// From apue 2
#include <stdio.h> {
#include <stdlib.h>
#include <pthread.h>

void *print_sth( void *ptr ){
}
char* message;
message = (char*)ptr;
printf("%s\n",message);
return NULL;
}
 int main()
{
         pthread_t thread1;
        char *message1 = "Thread 1";
        iret1 = pthread_create( &thread1, NULL, print_sth, (void*) message1);
        // Blocked. Until thread1 finishes
        pthread_join( thread1, NULL);
        printf("Thread 1 finished.\n");

        return 0;
}
```

**pthread_exit and pthread_cancel**

● <pthread.h>
● void pthread_exit(void *retval);
    ○ Terminate the calling thread and return the value
● int pthread_cancel(pthread_t thread)
    ○ Send a cancellation request to a threadProgramming Notification
● Include the corresponding header files
    ○ #include <pthread.h>
    ○ And others, ... e.g. "pthread_t" (id of a thread) is defined in
    <sys/types.h>
● Compile
    ○ $ gcc -Wall -o test pthread_test.c -lpthread
● Check the return value of a function call
    ○ Many functions write/record/set error information in a global variable errno when returning from errors
    ○ You can view the error message by examining errnoErrno, perror, strerror
● errno is an int variable declared in <errno.h>
● Each valid integer number (nonzero) represents an error code.
    ○ ($ man errno)

```
        2        ENOENT      No such file or directory
```

● A good programming style is to always check the return value of a function and the associated errno.
● perror() / strerror() can be used to translate the number stored in errno into a human-readable string, i.e., No such file or directory .
○ strerror() is more useful when you want to customize your own output format.

```
err = pthread_create(&ntid, NULL, thr_fn, "new thread: ");
if (err != 0) {
        fprintf(stderr, "can't create thread: %s\n", strerror(err));
        exit(1);
}
```

**Pthread Synchronization**

● Semaphores (#include <semaphore.h>)
    ○ Resources counter
    ○ Allows a limited number of threads in the critical section
● Mutexes
    ○ Binary semaphores
    ○ Allows only one thread in the critical section
● Condition Variables
    ○ Communicate information about the status of some shared dataMutex
● Header file: #include<pthread.h>

● pthread_mutex_init
● pthread_mutex_destroy
● pthread_mutex_lock
● pthread_mutex_unlockpthread_mutex_init
● Set the attribute of a mutex
● int pthread_mutex_init(pthread_mutex_t *restrict mutex, const pthread_mutexattr_t *restrict attr);
● We have three types of mutex
  ○ fast (default)
  ○ recursive
  ○ error checkpthread_mutex_lock
● If the mutex is unlocked: It becomes locked and owned by the calling thread
● If the mutex is locked by another thread, the calling thread is suspended until it is unlocked.
● What if the calling thread has already locked the mutex ?
  ○ pthread_mutex_lock(mutex);
  ○ pthread_mutex_lock(mutex);
  ○ ...
●If the calling thread has already locked the mutex
We have three types of mutex (statically defined below)
  ○ **fast (default)**
■ pthread_mutex_t fast = PTHREAD_MUTEX_INITIALIZER
■ Calling thread suspended on a locked mutex until it is unlocked (deadlock)
  ○ **recursive**
■ pthread_mutex_t recursive = PTHREAD_RECURSIVE_MUTEX_INITIALIZER_NP
■ Calling thread succeeds and returns immediately if a mutex is locked, recording the number of times it has locked the mutex. (Used when you unlock the mutex)
  ○ **error check**
■ pthread_mutex_t errchk = PTHREAD_ERRORCHECK_MUTEX_INITIALIZER_NP
■ Calling thread returns immediately if a mutex is locked

● Note: They are different only when the mutex is already locked by the calling thread.pthread_mutex_trylock
● It behaves identically to pthread_mutex_lock, except that it does not block the calling thread if the mutex is already locked by another thread(or by the calling thread in the case of a "fast" mutex)
● It returns immediately with error code "EBUSY".pthread_mutex_unlock
● Unlocks the mutex. The mutex is assumed to be locked and owned by the calling thread
● Be careful if the mutex is "recursive" or "error check" type. They will have different behaviors.
○ ***Refer to MAN page:***
■ http://linux.die.net/man/3/pthread_mutex_lockCondition Variable
● Header file: #include<pthread.h>
● pthread_cond_init
● pthread_cond_wait
● pthread_cond_broadcast
● pthread_cond_signal
● Note: Condition Variable will always be used with a mutexpthread_cond_init
● Unlike mutex, the attribute of a condition variable is
actually ignored. So "NULL" is always specified.
● Or it can be initialized statically,
  ○ pthread_cond_t cond = PTHREAD_COND_INITIALIZER;pthread_cond_signal
pthread_cond_broadcast
● pthread_cond_signal

    ○ Restart one of the threads waiting on the condition
    ○ If there are more than one threads waiting on the same cond, it is not specified which thread will be restarted.
● pthread_cond_broadcast
    ○ Restart all the threads waiting on the condition variablepthread_cond_wait
● Atomically unlocks the mutex (automatically)
● Waits for the cond to be signaled
● Thread is suspended and does not consume any CPU time until cond is signaled.
● Mutex must be locked by the calling thread before pthread_cond_wait()
● Before thread is recovered from the suspension at pthread_cond_wait(), it re-acquires (re-locks) mutex (automatically)pthread_cond_destroy
● Test whether there are threads waiting on the condition variable

**More helpful readings**
=> POSIX thread (pthread) libraries
=>Synchronizing Threads with POSIX Semaphores
=> POSIX Threads Programming
=> Multi-Threaded Programming With POSIX Threads
=>Multithreaded Programming (POSIX pthreads Tutorial)
=>Pthread Tutorial