

UNIVERSITY OF **ILLINOIS**

AT URBANA-CHAMPAIGN

CS241 System Programming

Discussion Section 1



www.uiuc.edu

MP Environment

- CSIL Linux Lab
 - 1245 DCL, 1265 DCL
 - Remotely accessible:
 - `ssh csil-linux[1-55].cs.uiuc.edu`
- For more information
 - <http://csil.cs.uiuc.edu>



Outline

- C Programming
- Makefile
- Debugger
- CS225 Review



C Programming

- C is the main language of Unix system programming.
- Why C in system programming?
 - Unix is implemented mostly in C.
 - Closer to machine
 - Optimization is easier



C vs. C++

- Similarities

- Built-in data types: `int`, `char`, etc.
- Preprocessors: `#include`
- Control structures: `if`, `for`, `while`, etc.
- Operators: `+`, `++`, `<`, `>`, `<>`, `=`, `==`, etc.
- There must be a function named `main()`.
- Function definitions
- Code can be split into each module and linked together when compiled



C vs. C++

- Differences
 - C does not have classes.
 - C does not have methods.
 - C does not support overloading.
 - In C, I/O is based on library functions
 - `printf`, `scanf`, `fopen`, `fclose`, `fwrite`, etc.
 - In C, dynamic memory allocation is based on library functions
 - `new` or `delete` are not supported.
 - `malloc()`, `free()`, etc.



C vs. C++

- Example (Linked List)

```
class Element {  
public:  
    Element(int item);  
    Element *next;  
    int item;  
};
```

```
class List {  
public:  
    List();  
    ~List();  
    void Append(Element item);  
    Element RemoveFirst();  
private:  
    Element *first;  
    Element *last;  
};
```

```
struct Element {  
    int item;  
    struct Element *next;  
};
```

```
Struct Element *List;
```

```
void Append(Element T, List L);  
Element RemoveFirst(List L);
```



C vs. C++

- Example (Linked List)

```
List L = new List();  
...  
delete L;
```

```
List L;  
L = malloc(sizeof(struct  
    Element));  
...  
free(L);
```



C Library

- Standard I/O (`stdio.h`)
 - `printf`, `scanf`, `fopen`, `fclose`, `fwrite`, etc.
- String functions (`string.h`)
 - `strcpy`, `strcmp`, `strtok`, etc.
- Standard library (`stdlib.h`)
 - `malloc`, `free`, `rand`, `srand`, `max`, `min`, etc.



Quick Glance at `printf`

- It is possible to print some values in the string. These values are determined in run-time.

```
printf("Hello World: %d.\n", 2006);
```

```
int i = 2006;
```

```
printf("Hello World: %d.\n", i);
```

```
printf("Hello World: %d.\n", i++);
```



Other Formats

- %d : signed integer
- %u : unsigned integer
- %x : print as hexadecimal
- %s : string of characters
- %c : single character
- %f : floating point number
- %% : print a single `\%`



C Library

- In CS241, you will learn how to use some of the following libraries.
 - Threads (pthread.h)
 - Signals (signal.h)
 - Timer (time.h, sys/time.h)
 - File Systems (fcntl.h)
 - Networking (sys/socket.h)
 - Misc. (unistd.h, sys/types.h)



References

- Check out the GNU gcc Manual online
 - <http://gcc.gnu.org/onlinedocs/gcc-4.0.2/gcc/>



Why Makefile

- Less time in compiling
- Easier to type 'make' than the appropriate gcc commands



Makefile Basics

- Type 'man make' and see make(1L)
- 'make' will parse and execute a file named Makefile.

- Makefile consists of
 - Variables
 - Labels
 - Comments: preceded by #



Variables

- Defining variables

```
CC = g++
```

```
THREAD_C = main.c \  
           scheduler.c \  
           system.c
```

- Using variables

```
$(CC) -c $(THREAD_C)
```



Labels

- Can be thought of as a subroutine

```
clean:
```

```
<tab>rm -o *.o
```

- Specifies dependency

```
main.o: main.h main.c
```

```
<tab>$(CC) -c $(CCOPTS) main.c
```

– Dependency can be another label.

- That label is executed first.



Example 1 (Bad)

all:

```
gcc -c -o main.o main.c header.h
```

```
gcc -c -o part1.o part1.c header.h
```

```
gcc -c -o part2.o part2.c header.h
```

```
gcc -o myprogram main.o part1.o part2.o
```



Example 2 (Good)

```
OBJECTS = main.o part1.o part2.o
```

```
HEADER = header.h
```

```
all: $(OBJECTS)
```

```
    gcc -o myprogram $(OBJECTS)
```

```
part1.o: part1.c $(HEADER)
```

```
    gcc -c -o part1.o part1.c
```

```
part2.o: part2.c $(HEADER)
```

```
    gcc -c -o part2.o part2.c
```

```
main.o: main.c $(HEADER)
```

```
    gcc -c -o main.o main.c
```

```
clean:
```

```
    rm -f myprogram $(OBJECTS)
```



References

- R&R p.807-809
- Check out man page
 - man make
- GNU Make Documentation
 - Link available in class website
 - <http://www.gnu.org/software/make/manual/make.html>



How to debug

- Use printf
- Debugger
 - GDB: GNU DeBugger
 - Most popular
 - DDD: Data Display Debugger
 - GUI version of GDB
 - Easier to use, but slow



GDB Overview

- GNU debugger
- The most popular debugger for UNIX systems
- A program that
 - runs other programs
 - allows the user to control and examine these programs



GDB Overview

- Four main functions
 - Start your program, specifying anything that might affect its behavior.
 - Make your program stop on specified conditions.
 - Examine what has happened, when your program has stopped.
 - Change things in your program, so you can experiment with correcting the effects of one bug and go on to learn about another.



Starting GDB

- First, compile your program with the “-g” flag (meaning “produce debugging information”)
- Three ways to start GDB
 - gdb program
 - gdb program core (useful after segfault)
 - gdb program pid (when debugging a running process)



Basic commands

- `run [arglist]`
 - Start your program (with `arglist`, if specified)
- `break [file:]function`
 - Set a breakpoint at function (in file)
- `bt`
 - Backtrace: display the program stack
- `print expr`
 - Display the value of an expression



Basic commands

- `c`
 - Continue running your program (after stopping, e.g. at a breakpoint)
- `next`
 - Execute next program line (after stopping); step over any function calls in the line
- `edit [file:]function`
 - look at the program line where it is presently stopped
 -



Basic commands

- `list [file:]function`
 - type the text of the program in the vicinity of where it is presently stopped
- `step`
 - Execute next program line (after stopping); step into any function calls in the line
- `help [name]`
 - Show information about GDB command name, or general information about using GDB



Basic commands

- quit
 - Exit from GDB



References

- R&R p.809-812
- Check out man page
 - man gdb
- GDB documentation
 - <http://www.gnu.org/software/gdb/documentation/>



Quick Review on CS225

- Linked List
 - Array
 - Stack
 - Queue
 - Priority Queue
-
- You should be familiar with all these data structures.



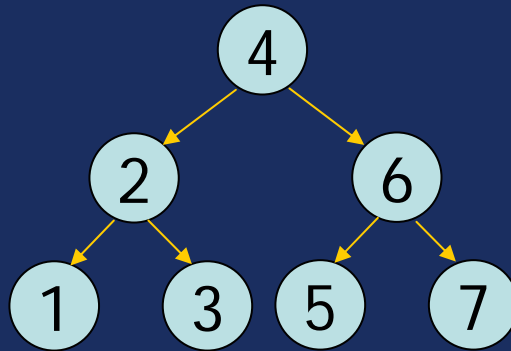
Array



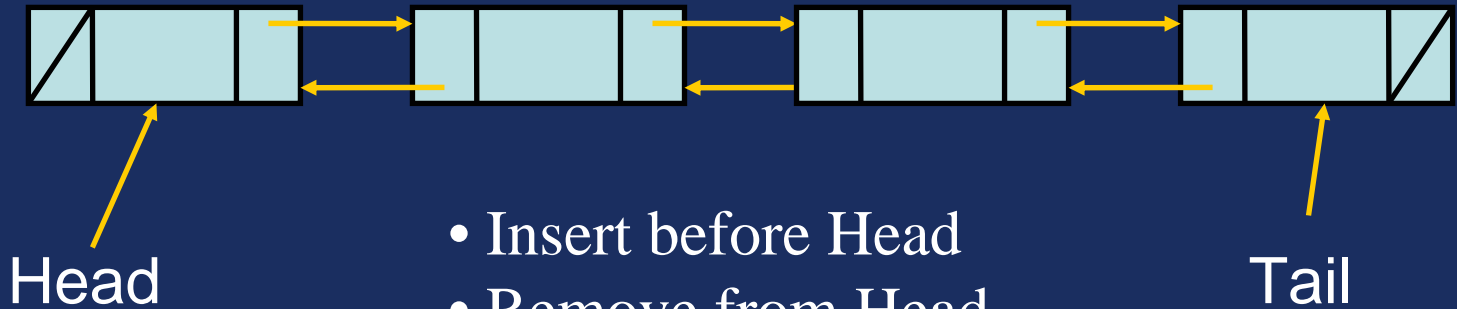
Doubly
Linked-list



Binary
Tree

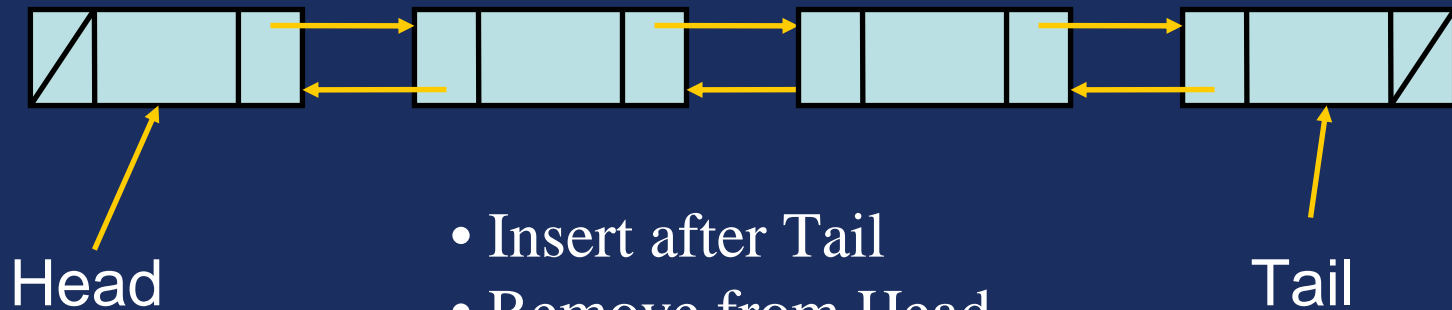


Stack



- Insert before Head
- Remove from Head

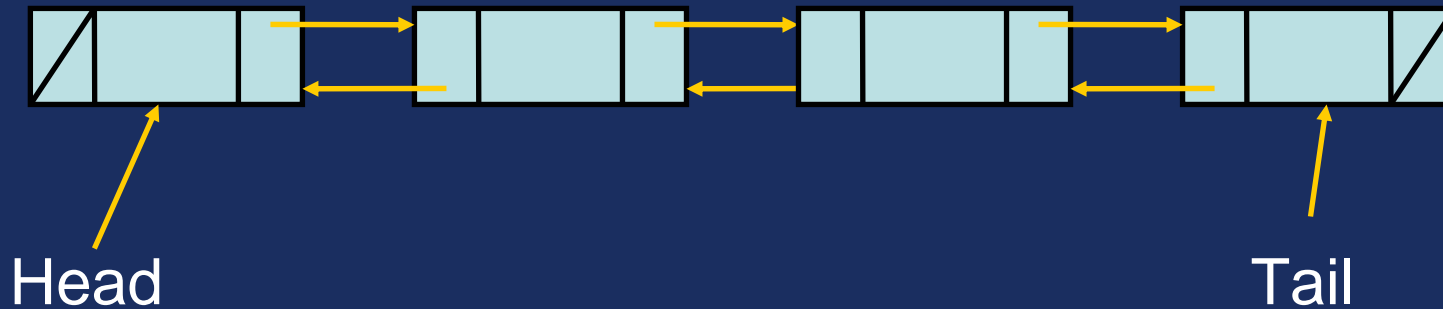
Queue



- Insert after Tail
- Remove from Head



Priority Queue (Sorted Linked-list)



- Insert – sorted by priority
 - Higher priority at Head
- Remove from Head

