Fedora 19
Fedora project.org
DVD Version

Fall 2013 CS288 Intensive Programming in Linux – Syllabus

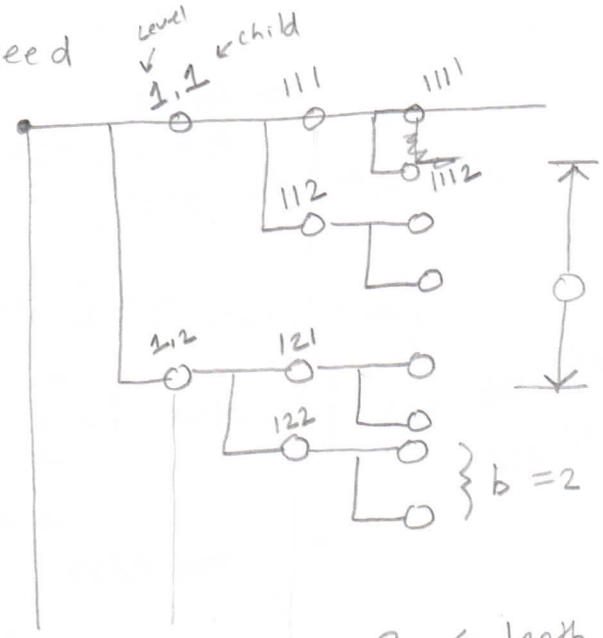
Check the announcements regularly. Class Web page: http://www.cs.njit.edu/~sohna/cs288/.

- Homework submission page: moodle.njit.edu
- Instructor: Andrew Sohn, GITC 4209, (973)596-2315, email: sohn -at- cs -dot- njit -dot- edu
- Office Hours: TBA and by appointment
- Teaching Assistant and office hours: Yang Li, yl98 -at- njit -dot- edu, time TBA, GITC 4217
- Class time and location: Tue,Fri, 1-2:25 pm, CKB 204, see the registrar page http://www.njit.edu/registrar/schedules/courses/fall/2013F.CS.html
- Textbook: The C Programming Language, Kernighan and Ritchie, Prentice Hall, 2nd ed., ISBN: 978-0131103627, and a Bash book.
- Recommended book (strongly recommended; read and understand before you graduate): Computer Organization and Design, Morgan Kaufmann, Fourth Edition, Patterson and Hennessy.
- Platform: Linux, distro Fedora 18 or above
- Tools: Bash, C, Python, mySql, DOM, PHP, some Java.
- Grading: 8 to 10 assignments (**0**%), two short *unannounced* quizzes (10%), Test 1 (20%), Test 2 (20%), Test 3 (20%), final exam (30%). Exam questions will be mostly from programming assignments. The three exams will replace three lectures.
- Exam dates will be announced in the first week of semester
- The NJIT Honor Code: You are required to read and follow the NJIT Honor Code.

Lecture schedule - Contents may change according to the class pace

1. **STAGE 1 - learning the most basic and fundamental**
   Intro to Linux, LAMP (Linux, Apache, mySql, Python/Php), virtualization
2. Intro to Bash shell scripting – variables, assignments
3. Intro to Bash shell scripting continues – arrays, lists, functions
4. Recursive directory traversal in Bash – depth first and breadth first
5. Pattern matching with regular expression (grep)
6. Introduction to C pointers, ref/dereferences,
7. Pointers to pointers, array of pointers, function pointers
8. Malloc/free and basic structure handling with simple linked list
9. Structure handling – swap and push
10. Structure handling with multiple links
11. **STAGE 2 - tools for building an end-to-end real-world application**
    Sorting - fast integer radix sort for integers
12. Sorting - introduction to floating point representation
13. Sorting - fast radix sort for floats
14. State space search – introduction, depth first, breadth first search
15. State space search - depth first, breadth first search
16. State space search - heuristic-based intelligent search
17. Matrix computation: a system of linear equation solvers
18. Matrix computation: introduction to iterative methods
19. **STAGE 3 - an end-to-end real-world application**
    Web processing - fetching with wget using Bash scripting, cron job, intro to DOM tree
20. Web processing - DOM properties, methods, and DOM tree navigation
21. Web processing - DOM properties, methods, and DOM tree navigation
22. Web processing - data extraction using Python minidom
23. Web processing - getting up and running mySql server, mySql DB construction, data injection
24. Web processing - getting up and running Apache server, reading DB using PHP, constructing clickable/sortable
25. Web processing - plotting charts and graphs and presenting data using PHP
26. **STAGE 4 - extending tools and applications to run on many-core machines**
    Introduction to multicore/parallel computing using MPI (optional OpenMP) - point to point communication
27. Introduction to MPI (optional OpenMP) – collective group communication
28. Simple matrix computation for multicore/multiple machines using MPI (optional parallel radix sort using MPI on a cluster of machines)

b = 2 d = 3




```
cd $seed_d
mkdir 11
mkdir 12

cd 11
mkdir 111
mkdir 112

cd 12
mkdir 121
mkdir 122
```

-v (verbose)

level = 1

```
for ((i=1; i<=b; i++)); do
    echo "mkdir ${level} ${i}"
done
```

Use inner loop to create d and b

Making directories in shell

open – dir going to create
closed – dir created
temp –

```
open = (seed)
for ((i=1; i<depth; i++)); do
    new_open = ( )
    for x in ${open}; do
        temp = ( )
        for ((j=1; j<=b; j++)); do
            d = $x/$j
            mkdir $d
            temp = ($temp $d)
        done
        temp = (11, 12)
        new_open = ($new_open $temp)
done
```

1st
open = (seed)
closed = ( )
temp = ( )

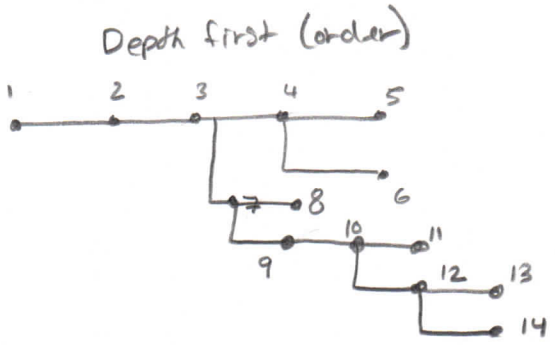
2nd
open = (11 12)
closed = (seed)
temp = ( )

3rd (not explored/expanded)
open = (11, 111, 112, 12)
closed = (seed, 11) ← used to expand
temp = (111, 112) ← successors

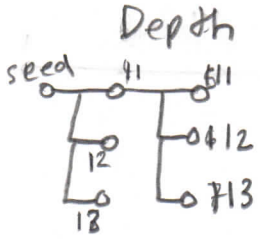
9/23

Exam Friday study HW1 and maybe HW2?

Traverse Tree: Breadth first, depth first; recursively and



Look up breadth first vs depth first



Depth 10, Breadth 10 → 10 (Breadth) 10 (Depth) → number of created directories

Backreference:

Match abc → abc
xyz → xyz

abc.*abc.*xyz.*xyz
\1 ← Label 1 \2 ← Label 2

abc xyz abc xyz
\1 \2

Mr␣\([a-z]*)␣came
\1 ... \1

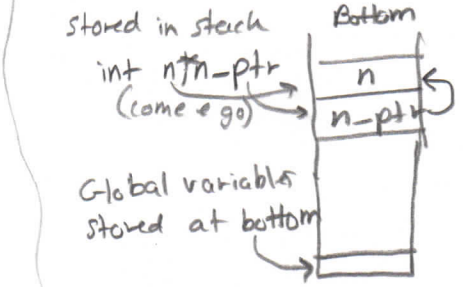
Mrs␣\1

* C pointers: http://cslibrary.stanford.edu/102
*Read & Study

n = 42
n_ptr = &n;
↑ Location of n; memory address of n

*How to use mkfile

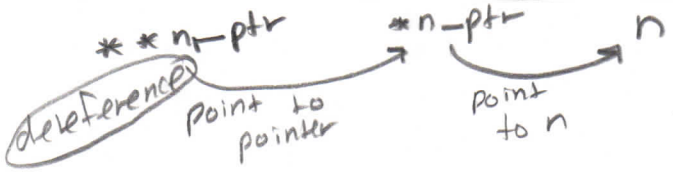
* 40 – 1TB
32 – 4GB
31 – 2GB
30 – 1GB
20 – 1MB
10 – 1k



n_ptr holds address of n

*n_ptr → dereference n_ptr

&n_ptr → reference n_ptr



Subject: pointers

Date: 10/1

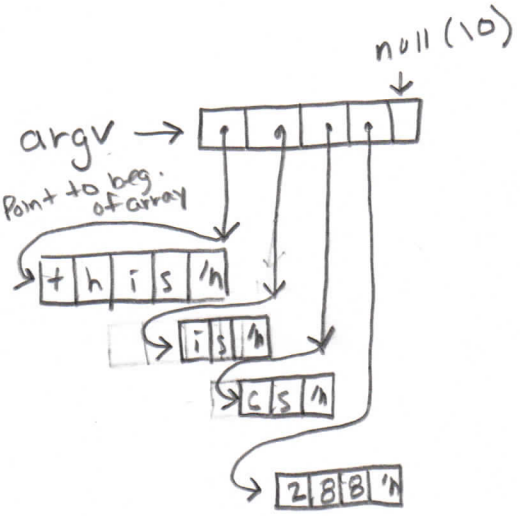
hello.c

```
#include <stdio.h>
int main (int argc, char **argv){
    printf("this is cs 288 \n");
}
```

int argc → simple integers; char **argv → string of char; pointer to pointers

$ hello this is cs 288

Arguments: 4 parameters

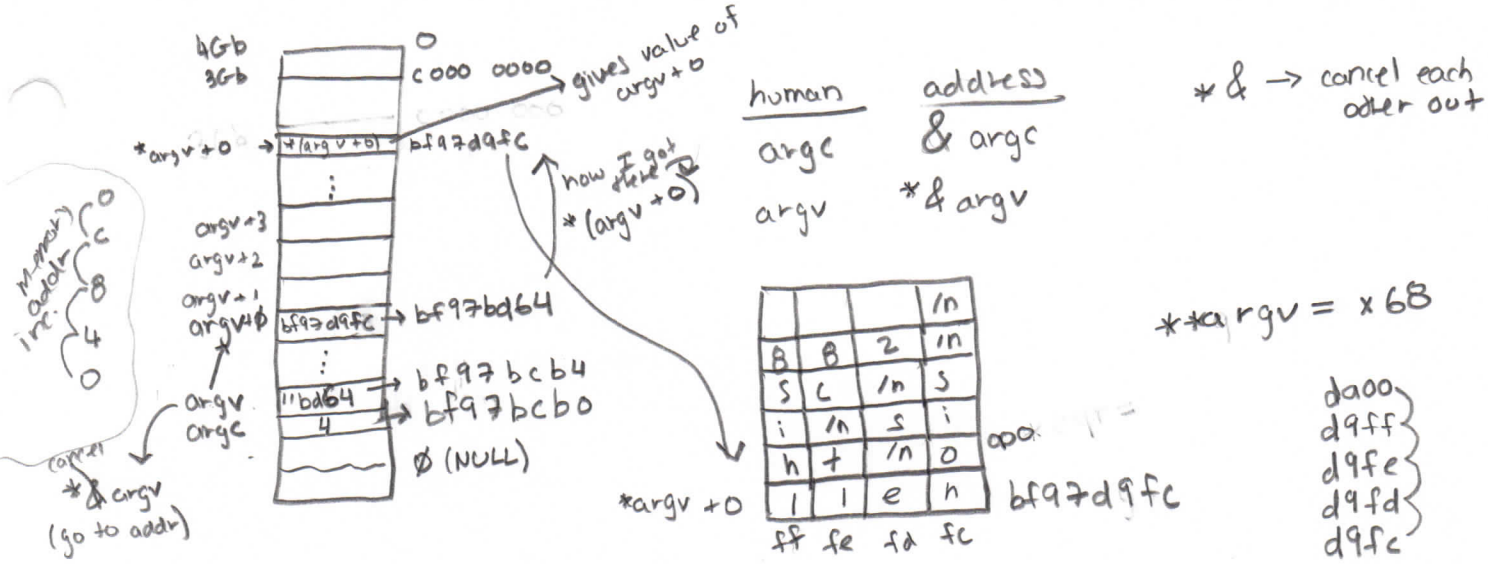


makefile

```
hello: \t hello.o
hello.o : cc ...
```

argc → argument count

argv → argument vector



human	address
argc	& argc
argv	*& argv

*& → cancel each other out

**argv = x68

Subject: Linked-Lists Date: 10/4

① Declare var

```
struct node {
    int data = 1;

    struct node *next;

};

struct node *hp;
```

```
hp -> data = 1;
hp -> next = NULL;

second -> data = 2;
second -> next = NULL;
hp -> next = second;
```

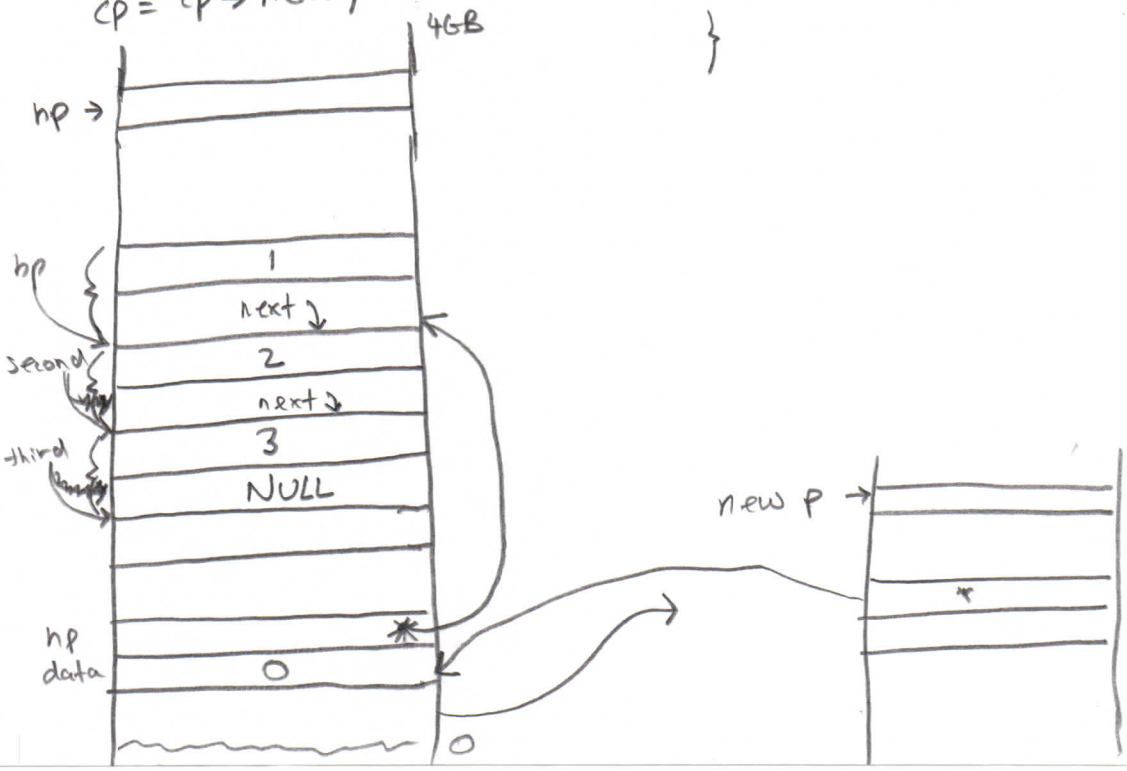
② init

```
hp = (struct node *) malloc (sizeof (struct node))
    struct node *second;
second = (struct node *) malloc (sizeof (struct node))
    struct node * third;
third = (struct node *) malloc (sizeof (struct node))
    third -> data = 3;
    third -> next = NULL;
    second -> next = third;
```

loop to traverse & count LL;

```
int struc_len (struct node *hp)
int cnt;
struct node *cp;
cnt = 0;
cp = hp;
while (cp != NULL){
    cnt++;
    cp = cp -> next;
}
```

```
struct node *cp;
cp = hp;
cp = cp -> next;
```

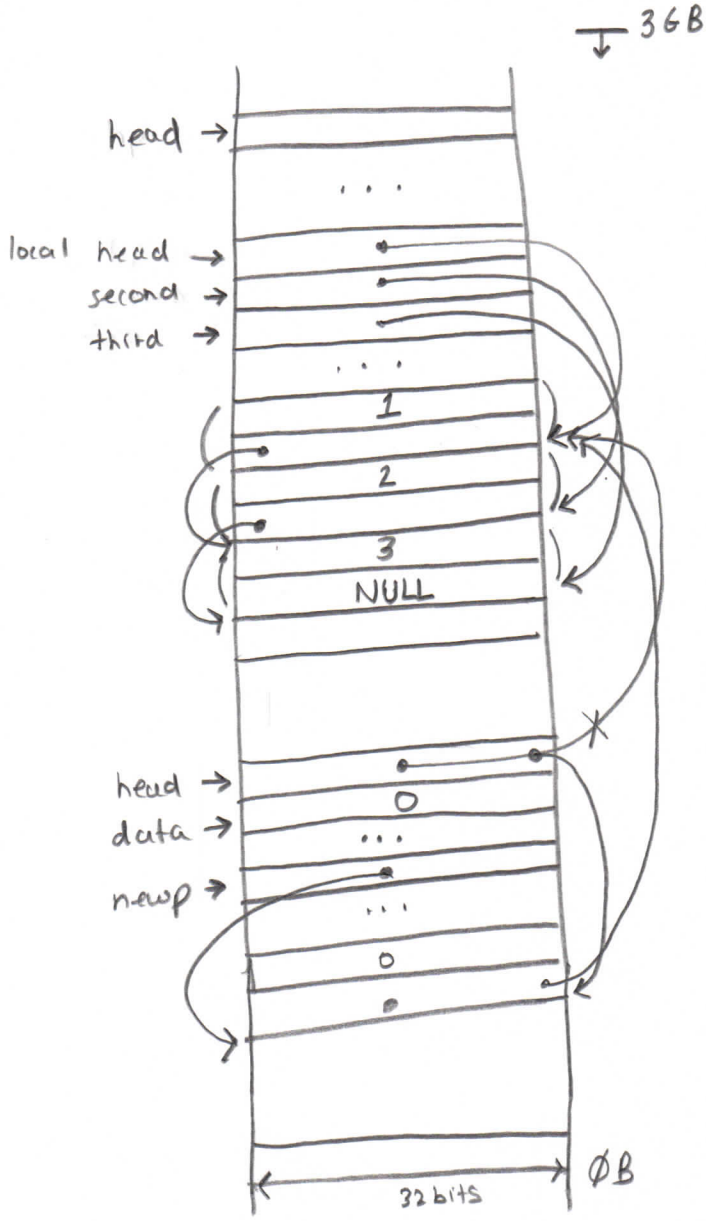


Subject: C Pointers Date: 10/8

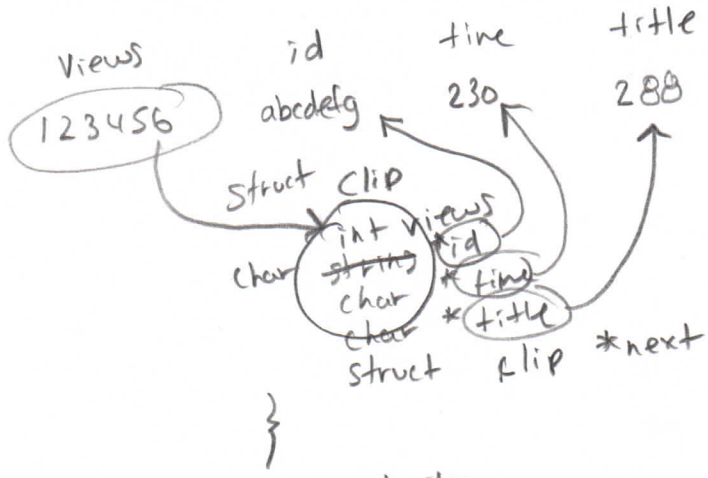
```
struct node {
4B  →  int data;
4B  →  struct node *next;
    }
```

malloc ? = size of (struct node)

= 8 byte



Subject: Views Date: 10/8



}

- strtok()

```
strtok(s, ";");
```

(delimiter: ";", s: string)

First call:

① look for the first not on the delims
② look for the first on the delims set that to NULL

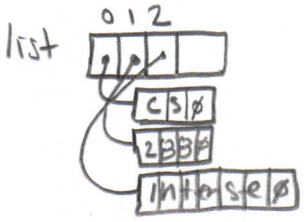
```
strtok(NULL, ";");
```

"CS␣ 288␣ Intensive␣ Program"

```
token = strtok(s, "␣");   // = CS
  1.
  2.
token = strtok(NULL, "␣");   // = 288
  1.
  2.
```

* Need a loop for string of n-length

```
char *token;
token = strtok(s, "␣");
while ( token != NULL) {
    list[i] = malloc (strlen(token))
              strcpy(list[i], token)
    token = strtok(NULL, "␣");
    i++;
}
```



} "CS␣288␣Intense"

Subject: ~~Selection~~ Sorts Date:

Ex

Selection sort

55 10 32 7 9 21

↑ find smallest and swap

7 10 32 ~~55~~ 9 21 n^2

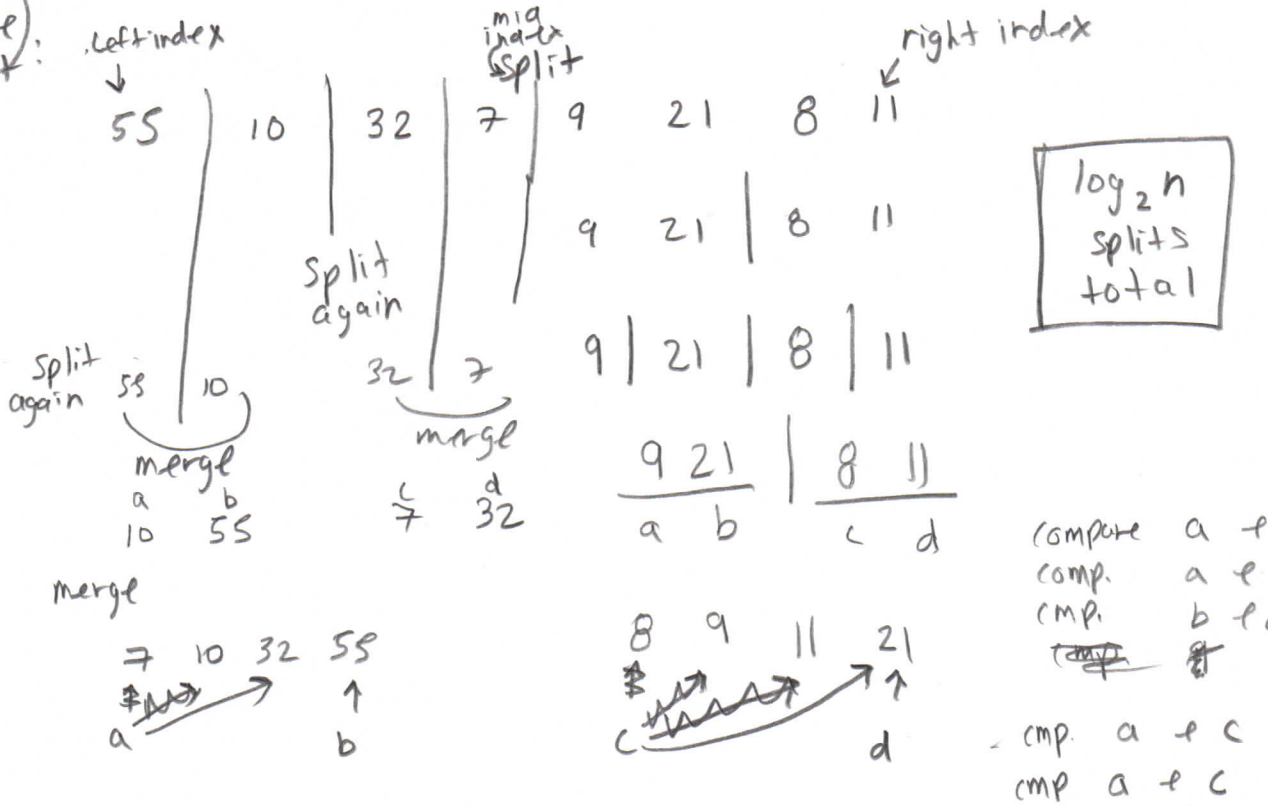
7 9 32 55 10 21

7 9 10 55 32 21

7 9 10 21 32 55

7 9 10 21 32 55 DONE

Merge sort:



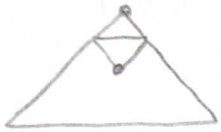
$\log_2 n$ splits total

7 8 9 10 11 21 32 55

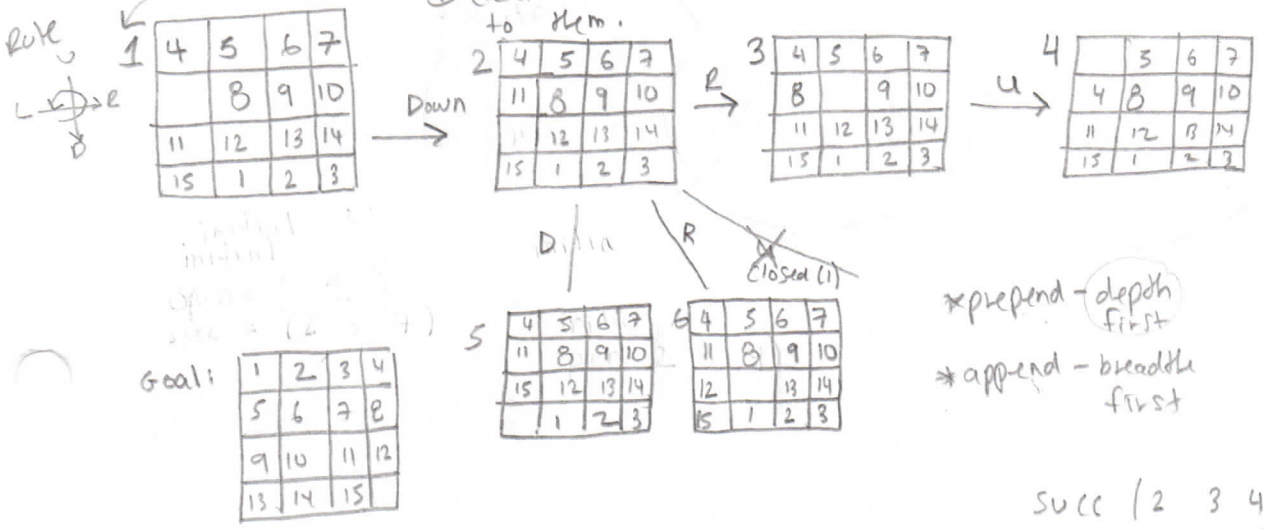
Subject: Searches

Date: 10/22

* blind search (brute force)
  - depth first 2^{50-60}
  - breadth first
* intelligent search
  - best first 2^{30}
  - branch and bound
  - a*
* fancy / sophisticated
  - Bidirectional 2^{10}
  - multidirectional



① create rules and stick to them.



initial
open (1)
succ (2 3 4)
closed (1)

open (2 3 4)

closed (1)

succ = (5 6 ~~7~~)

succ (2 3 4 5 6)

* prepend - depth first
* append - breadth first

Goal:

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

open depth (5 6 3 4)

open breadth (3 4 5 6)

open best first (6 3 5 4)

```
struct node *open, *closed, *succ;
            *goal;
struct node {

    int state [4][4];

    struct node *next;
                *prev;

};
```

```
init
while (                    ){
    - pick from open (the first)
    - add to closed
    - expand = succ
    - check (succ, goal) → if true, break
    - check (succ, closed)
    - merge (succ, open, flag)
        ↳ 1 = depth
          2 = breadth
          3 = branch
          etc.
}
clean-up
```

Design
Develop
Implement
Debug

Subject: Brute force Search

Try circular linked list

Date:

```
open = (initial)
while ( 1 ) {
  cp = initial
  succ = expand (cp)
  if goal_found (succ)
      break;
  succ = check (succ, open)
  succ = check (succ, closed)
  closed = merge (cp, closed)
  open = merge (succ, open)
}
```

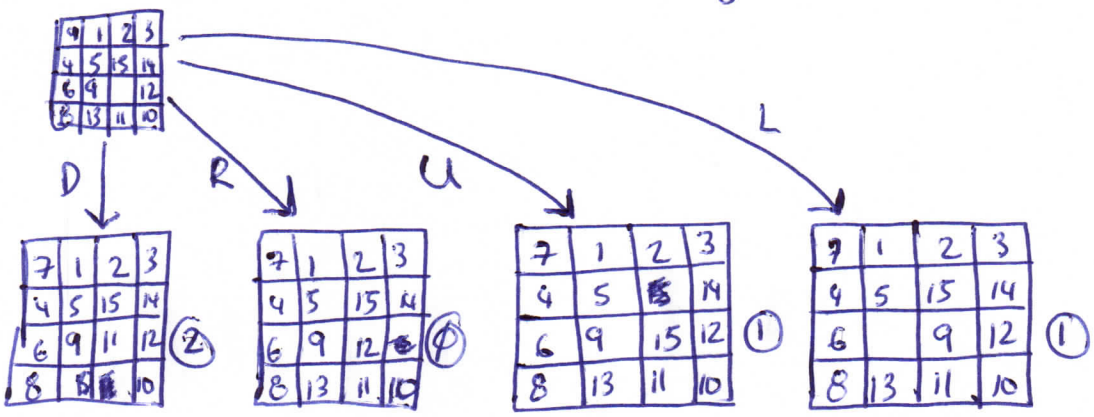
```
int check_move ( x , y )
  int flag = 0
  if (x ≥ 0 AND x < N)
    flag = 1;
    AND (y ≥ 0 AND y < N)

check (succ, open)
```

```
struct node {
   int board [4][4]
   struct node *prev, *next;
}
```

```
find_blank ( b)
for (i = 0; i < N; i++)
  for (j=0 ; j < N; j++)
    if (cp → board [i][j] == ∅)
        break;
flag = check_move (i+1, j)
    if (flag)
tp = malloc (size of)
```

Subject: h = heuristic Best First Search Date:

estimated distance to the goal



Subject: Date:

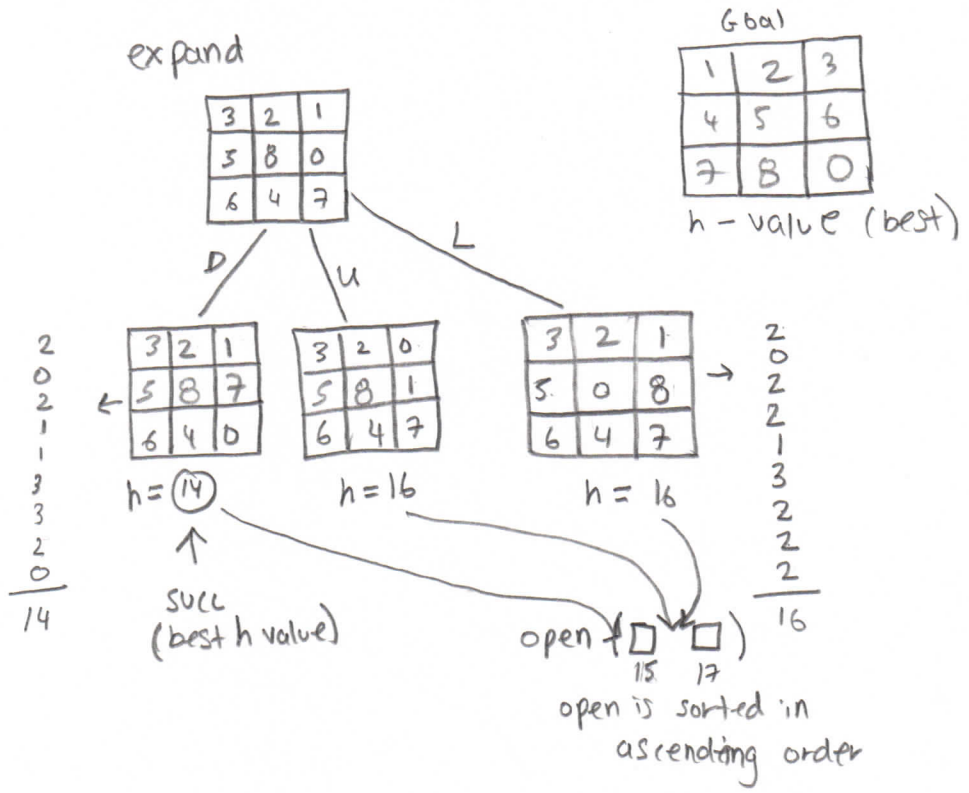
move function

filter function is a comparison.

```
struct node *move (int i, int j, int x, int y, struct node *cp)
    struct node *tp;
    tp = NULL;
    if (x >= 0 && x < N && y >= 0 && y < N) {
        tp = malloc(
        for (i = 0 ...)
            for (j = 0 ...)
            tp -> loc[i][j] = cp -> loc[i][j]
            swap (i, j, x, y, tp -> loc)
    }
return tp
```

Subject: Best First Search
Manhattan Distance

Date:



g = distance travelled from parent
h = estimated steps from goal

$f = g + h$ → A*

Bidirectional Search (Dont need to know)

Subject: **Date:**

initial

fif 1 2 3 ... 67 dfs

argv {0}

```
for (i = 0; i < N; i++)
  for (j = 0; j < N; j++)        Atoi
    initial -> loc[i][j] = (argv[k])
```

Expand() function

```
expand (struct node *tp)
   tp -> loc[][]
   for (i=0; i < N; i++)
     for (j=0 ; " ")
       if (loc[i][j] == 0)
          break;
   if (sp = move (i,j,i+1,j, tp))
       merge (sp, succ)


return succ
```

Subject: Parallel Programming **Date:** 12/2

① Log into another computer without password:
- ssh keygen -type rsa
  (passwordless login)
- disable firewall

- append public to authorized keys
  cat rsa public (if you have more than 3 machines) you have to append

100 - au_k pub 1, pub 2 append keys

101 - au_k pub 0, pub 2 append keys

102 - au_k pub 0, pub 1 append keys

cat >> authorized_keys (will append the keys instead of overriding the keys)

② Have MPI installed (open MPI)

openmpi and openmpi_devel ← Packages

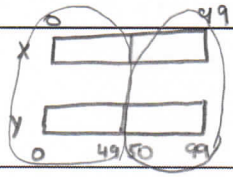
LD_LIBRARY_PATH = $LD_LIBRARY_PATH:/usr/lib/openmpi/lib

https://computing.llnl.gov/tutorials/mpi
lbl.gov

cat hosts

Subject:

dot 1000



Date:

dot_product
- nprcs (# processes)
- pid (processor id)
- n (problem size)

s_idx = work * pid
work = n/nprcs
e_idx = s_idx + work

dot_product (s_idx, e_idx, x, y)

my-prod = dot_product (s_idx, e_idx, x, y)

int prod = my_prod

master ____ worker
collect

pid=0: recv

for (i=1; i<nprcs; i++)
prod = prod + recv ( )

pid 1: send

• If not master:
(my_prod)
send 1 of type integer
to master.

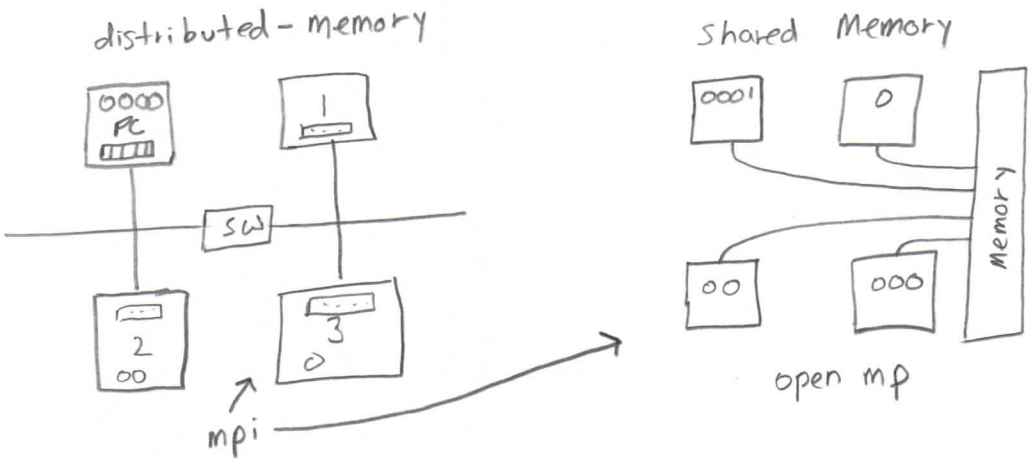
mpi_send (my_prod, 1, int,
master, 1234, world)
to — color? matches with recv — ourselves only

If (pid == master)
for (i=1; i<nprcs; i++)
prod = prod + MPI_Recv (1, int, i, 1234, world)

values, type, matches send, in this "network"

Subject: **Date:**

MPI
- Compile MPI → mpicc
- Run MPI → mpirun



MPI_Gather

MPI_Barrier ← Prevents actions from occurring until finished

MPI_scatter

MPI_Broadcast

MPI_Scatter (A, work, int, A, work, int, master, world)

MPI_Bcast
(B, n×n, int, master, world)

for i
for j
for k
} dot-product

MPI_Gather
(C, work, int, c + w* id,

Subject: mpi apr in caps **Date:**

```
int main ( int argc, char *argv[])
    int num_procs, rank, namelength
    char processor_name [mpi_max_processor_name]

    mpi_init (&argc, &argv)
    mpi_comm_size (mpi_comm_world, &numprocs)
    mpi_comm_rank ( mpi_comm_world, &rank)
    mpi_get_processor_name (processor_name, &namelength)

    if (rank == 0)
        printf ("[%d/%d %s): master %d: Hello\n", rank+1, num_procs,
            processor_name, rank);

    else
        printf ("(%d /%d %s): worker %d: world!\n", rank+1, num_procs,
            processor_name, rank);

    MPI_Finalize ()

#include <mpi.h>
#include <unistd.h>
#include <stdio.h>
```