CS 242 notes

Point3D point;
point.data[0] = 1.0;
char str[10], c = 'a';
#include <cmath>

sqrt  
abs, fabs

# include "test.h"
# define sin cos
f.open("filename", ios::out | ios::app)

throttle =
flow();  // call flow of instance

throttle::flow();  // call static function flow
char string[20] = "Hello world";
for (char c : string) {
    cout << c << endl;
}
char x = string;
for (char c : x) {
    cout << c << endl;
}
char x = "Hello world";
char x = new char[128];
int a, b;

b = 5;

char * c = new string(“hello”);
char * c = “hello”, new char[12];
char * c = new char[12];
c = "Hello World"
str = "Hello World"
strcpy (c, "Hello World");
strcpy (str, "Hello World");
char str[9] = "Hello World";
char c = new char[128],
    x = new int;
int y = new int;
delete c;
delete x;
delete y;
\log (\ldots)
\
\{ \\
\ln x; \\
\ln x = \ln u + \ln i;
\}

\[ch_5 \neq c, \]
\[ch_\alpha \neq b; \]
\[ch_\alpha, \neq b; \]
int x i = new int;
(= int x i = new int[1]);
i[0] = 5;
(= x i = 5);
char str[128];
```c
int a;
int x;
ν = &a

if (x == a)
```
call by value:
int test (int a);
call by reference:
int test (int &a);
int test(int a);
int test(int b, c),
int b;
int a;
b = a;
increment/decrement hw.

```c
int i;
for (i = 0; i < 10; i++) {...

i++;
c++;
```
int array[100],
int *p;

p = array;
p++; \implies p = \& (array + 1);
p++;
Point3D array[100];
Point3D xp = array;
p++;
$p = i$;
$(p - array)$
int *array,
array = new int[0000000],

void test (int &x);
void test (const int &x);
void test (int x, const Point3D& point, point.length());
linked lists

head → 10 → 2 → 15 → tail

4 → 10
head = 0

singleNode *pointer;
polygon->next;
(\*pointer).next;

int *data; \< in form of data

int \*p = data;

data = new int [5];

for (i=0 .. j)

data[i] = p[i];


```c
delete p;

usage of *

int *data; // create a pointer to int

int val;   // create an integer variable
```
*data = vec; data = new int;
*data = 5

vec

5
\texttt{data} = \&\texttt{val},
```c
int data[50],
int *p = & (data [5]);

data [5] = 5; p[0] = 7;
p[5] = 5;
```
\texttt{\textit{data} = \textit{data} \oplus 0} \\

\begin{verbatim}
struct single
{
    data;
    next;
};
\end{verbatim}

\begin{verbatim}
single \* pointer;
pointer \* next = 0;
\end{verbatim}
(pointer), next

int *getData () {
    return p;  // Here p is of type int
}

int *getData () {
return b;
}
creating a temporary pointer for deleting an element:

```
pointer
```

```
[ tmp ]
```

```
data
next
```
inserting an element:

```
data
next

data
next

[??]

data
next
```
adding at the end of the list:
Deleting the last element:

```
data
next

```

```
data
next → 0
```

---
singly linked list:

head -> [data] -> [next]

doubly linked list:

head -> [data] -> [next] -> [prev] -> [data]
Inserting an element into a doubly linked list

```
data
next
prev
```

```
data
next
prev
```

```
data
next
prev
```

Pointer
Inheritance:

Dose class

special class

primitive

triangle

rectangle
virtual function
at runtime program would
check type of class and call
its version of that function.

Pure virtual functions:
only provide declaration,
but not the actualimple.
match him

iterator:
  iterator through the list
  - begin
  - end
- ++ operator
- * operator
- = operator
- == operator
- != operator
Stacks:

\[ \text{pop} \rightarrow \quad \]

\[ \equiv \rightarrow \quad \equiv \]
push

\[
\begin{array}{c}
\text{class lists,} \\
\text{typename lists.}
\end{array}
\]
Queues

stack: LIFO
(last in, first out)

queue: FIFO
(first in, first out)
Two approaches:
- limited number of entries
- unlimited elements

[first last]
Trees

node:
- data
- child
- child?
pre-order
in-order
post-order
Towers of Hanoi
Rock tracking

divide and conquer
Searching

10 3 9 2 5 3 8 11 20

linear search

binary search tree
binary search

Sorting:

List: insert element in a sorted fashion
$O(n^2)$

Bubble sort:

Complexity $O(n^2)$

5 1 4 8 2

-> 1 4 5 8 2
complexity: $O(n^2)$

performs well on (almost) sorted lists or arrays

merge sort

$$2^x = n \Rightarrow \log_2 n = x$$

$2 1 5 1 9 11 11 4 7 6 18 2 12 11$

$1 1 2 4 10 19 11 11 6 10 7 18 11$
QuickSort

Complexity: worst case: $O(n^2)$
average case: $O(n \log n)$

Complexity: number of elements

$O(n^2) \Rightarrow m \cdot n + c$ operations

$O(n) \Rightarrow m \cdot a + c$

Heap sort
Heap

Binary search but with different sorting:
- parent element is always larger than its children
element i: children 2*i + 1, 2*i + 2

from child to parent: (i-1)/2

second case for sorting:
exange first and last element:
10 22 / 27 / 5 / 45

27 22 10 5 45

\[ 27 \rightarrow 22 \rightarrow 10 \rightarrow 5 \rightarrow 45 \]
after swapping:

\[
\begin{array}{cccc}
5 & 12 & 10 & 27 & 40
\end{array}
\]

\[
22 \quad 10
\text{    } \Rightarrow \text{    }
\]

\[
\begin{array}{cccc}
22 & 5 & 10 & 27 & 40
\end{array}
\]
Move largest element to the end

| 10 | 5 | 22 | 27 | 45 |

<table>
<thead>
<tr>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
</tr>
</tbody>
</table>

| 5 | 10 | 22 | 27 | 45 |
Initial computation of heap

\[ \begin{align*}
\text{heap} & \quad \rightarrow \\
5 & \quad 7 \\
10 & \quad 22 \\
\text{heap} & \quad \rightarrow \\
27 & \\
45 & \\
\text{heap} & \quad \rightarrow \\
5 & \\
10 & \\
\end{align*} \]
Complexity:

$O(n \log n) + O(n \cdot \log n)$

Initial heap
Finding largest element
and restoring heap

Hence: overall complexity: $O(n \log n)$