From Page No.__

Outline

- Across layers : ~~Storage~~ perspective
- Objects → ptrs/structs → memory
- Dynamic allocation
- Lower level allocation

Administration
- Project
- midterm
- Review
- Labs

I Goal of CS61C : "Machine Structures" :

allow you to work and think within

and across layers of abstraction

objects, data structures

{ ~~Syllabus~~ — HLL Program (JAVA, C)

P.L Runtime — heap, stack, ~~code~~ DATA, code

organize the basic machine capabilities to support language concepts

Machine

- fetch inst
- fetch operands
- compute result
- store
- next inst

- information
- storage
- processing

Memory
- sequence of words
- Address
- Value (bits)
  - Data
  - instructions

To Page No.____

Witnessed & Understood by me,	Date	Invented by	Date
		Recorded by	

From Page No.

II ~~Hit~~ JAVA → C → Machine

JAVA

```
public class Point {
   public static int npoints = 0;
   public int x, y                    <= public Point next;

   public Point(int x, int y) {
      this.x = x;
      this.y = y;        npoints++;      => next = NULL
   }

   public Move (int dx, int dy) {
      x = x + dx;
      y = y + dy;
   }

   protected void finalize() {
      npoints--;
   }

}

{
   Point P;          declare an object
   Point Q = new Point(2, 3);
   Point R = P.

   P.move(3, 4);
```

public connect (Point Q)
this.next = Q

To Page No.

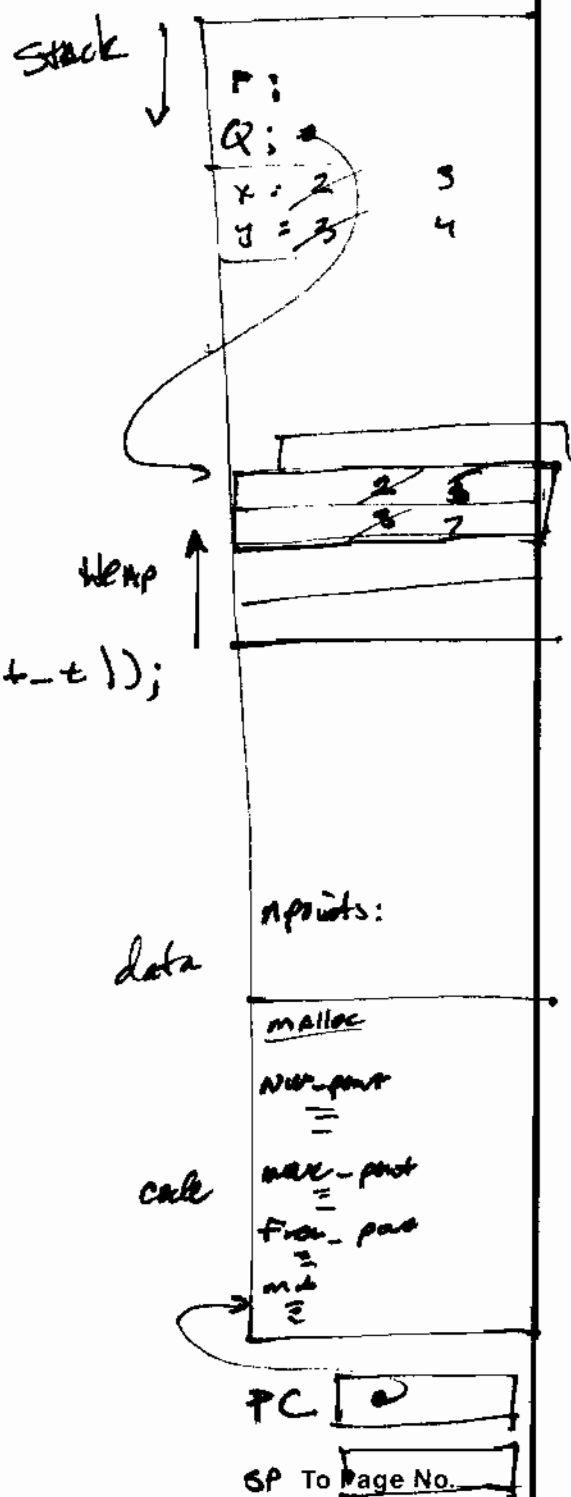
Witnessed & Understood by me,	Date	Invented by	Date
		Recorded by	

C

Machine

~~Point~~.c

```
static int npoints = 0

typedef struct Point {
    int x, y;
} Point_t;

typedef Point_t *Point;

struct point *new_Point(int x, int y) {
    struct point *np = malloc ( sizeof ( Point_t ));
    np -> x = x;
    np -> y = y;        npoints ++;
    return np;
}

                    Point P
void Move_Point ( int dx, int dy ) {
    P -> x += dx;
    P -> y += dy;
}

void free_Point ( Point P ) {
    free ( P );         npoints --;
}

main {
    Point P;
    Point Q = new_Point ( 2, 3 )
    P = Q;
    Move_Point ( P, 3, 4 )
```



Witnessed & Understood by	Date	Invented by	Date
3		Recorded by	

From Page No.__

~~what if too~~ In else

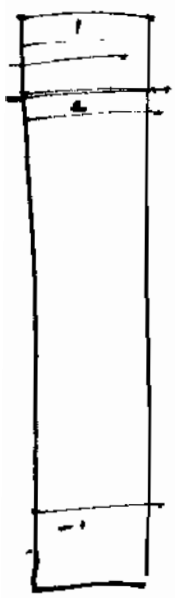
How does dynamic memory manage work?

Example: Store of points

```
int ~~Point~~ ~~free_point~~ = ~~NULL~~ -1
    Point_T Points[MAXpoints]

struct Point {
    int next;
    int x;
    int y;
}

void init_Point( ) {
    int i;
    for (i=0; i < MAX Points; i++) Points[i].next = i+1
    Points[ -1 ].next = NULL;
    Free
~~Point new_Point(int x, int y)~~
}

~~P~~
int new_Point(int x, int y) {
    ~~if free~~
    int np;
    if (free_point >= 0) {
        np = free_point
        free_point = points[freepoint].next
        points[NP].next = -1
        return np;
    } else {
        fail ("No more points")
    }

void free_Point(int p) {
    points[p].next = free_point;
    free_point = p;
}
```

To Page No.__

Witnessed & Understood by me,	Date	Invented by	Date
		Recorded by	

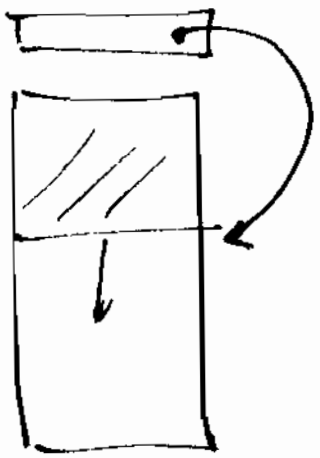


```
typedef struct point {
    struct point *next;
    int x; int y;
} Point_t;

Point free_point = NULL;
point_t Points [MAXPoints];


void init_Points () {
  ~~Point p~~ int i;
  for (i=0; i < MAXpoints-1; i++)        <- Iterate over points?
      Points [i]. next = & Points [i+1];
  Points[i]. next = NULL
}


int new_Point (int x, int y) {

  Point np;
  if (Free*Point != NULL) {
     NP = free_Point;
     free_point = Points [free Point ]. Next;
     ~~Point~~ NP -> Next = NULL
     return NP;
  } else {
      fail
  }

void free_point (Point P) {
    P-> Next = free_point;
    free_point = P
}
```


Witnessed & Understood by me,	Date	Invented by	Date
		Recorded by	

From Page No.

A stack of "anything"



stack

```
int sp = 0;
char stack[MAXSTACK];

char *alloc(int len) {
  int nsp = sp + len;
  if (nsp >= MAXSTACK) return NULL;
  sp = sp + len;
  return &stack[nsp];
}

void *free(char *p, int len) {
/* Check? */
  int nsp = p - stack;
  if
}

int *x = alloc(sizeof(int));
Point p = alloc(sizeof(Point));
free(p);
free(x);
```

To Page No.

Witnessed & Understood by me,	Date	Invented by	Date
		Recorded by	

```
char stack[MAXSTACK];
char *SP = stack;


char *alloc(int len)
  int NSP;
  if (SP+len >> &stack + MAXSTACK) return NULL
  NSP = SP
  SP = SP+len
  return NSP
}
```

How could we check that we are the last thing allocated?

- Mauler

Witnessed & Understood by me.	Date	Invented by	Date
		Recorded by	