Today

1. Review
2. Dynamic Memory Allocation
3. Linked Lists
4. File I/O

Array

```
a: 0  1  2  3  4
  1271 -123  0  71  -1712
  0x1000 0x1004 0x1008 0x100c 0x1010
```

```c
int a[5];

sizef(int) = 4 bytes
```

double d[5];

```
double) = 8 bytes
```

Pointers

```
int num = 10;
int * p;
p = &num;
*p = 20;
main() {
    int num = 10;
    addOne(num);
    print num;
}
void addOne(int num) {
    num = num + 1;
}
```
void addOne(int *pnum)
{
    *pnum = *pnum + 1;
}

In main()
addOne(&num);

int **pp = &p;

**pp = 20;

LIFO

4
3
2
1

phum

0x1000

num

0x1000

0x2000

0x3000

num

16 20

p

0x1000

pp

0x2000

0x3000
Arrays of structures

struct point pt[5];

pt[i].x = value;

Self referential structures

head

struct node {
    int data;
    struct node *next;
};

struct node {
    int data;
    struct node *child;
};

n
y

p[0]
p[1]

x
d
y

data child

10 20

4 bytes

4 bytes
int global = 1;

main () {
  -> int local = 0;
  -> fn (...);
}

global = 2;

fn (...) 

(global) stack 

main: local 0

code

data

heap

global variables

local variables
Dynamic Memory Allocation.

Get memory from heap.

`void * malloc (size_t size);`

```
void * p = malloc (8);
int * p = malloc (8);
```

Int * p = malloc (8);

Contiguous region of memory

```
OX2000
```

```
4 bytes

OX DD
OX DD
OX DD
OX DD
OX 01
OX DD
OX DD
OX DD
OX DD
OX DD
```

```
OX2004
```

```
* (p+1) = 2;
* (p+0) = 1;
* (p+0) = 1;
* (p+1) = 1;
```

```
OX2000 + 1 * sizeof(int)
```

```
OX2007
```
The diagram illustrates the allocation of memory for an array of integers in C. Here's the explanation in plain text:

- `int`: 4 bytes, 32 bits

- `malloc(10)`: Allocates 10 integers

- `int *p = malloc(sizeof(int) * 10);`: Allocates memory for a array of 10 integers

- `p = malloc(sizeof(int) * 1000);`: Attempts to allocate more memory

- If `p` is `NULL`, it indicates that the `malloc` call failed, and there is no more memory available in the heap.
Need more memory?

Stack

Heap

\[ p[1] = 7; \]

\[ p = \text{malloc} \left( \text{sizeof} \left( \text{int} \right) \times 20 \right); \]

\[ \text{free} \left( p \right); \]
```c
int *p = malloc (8);
p = NULL; // memory leak!
free(p);
free(NULL); // no op.
```

```
struct node { 
    int data;
    struct node *next;
};
```
main ( ) {

1) struct node *head = NULL;

2) struct head = malloc (sizeof (struct node));
   head -> data = 10;
   head -> next = NULL;

}
```c
int delete (struct node **phead) {
    assert (first != NULL);
    struct node *first = *phead;
    int data = first->data;
    *phead = first->next;
    free(first);
    return data;
}
```