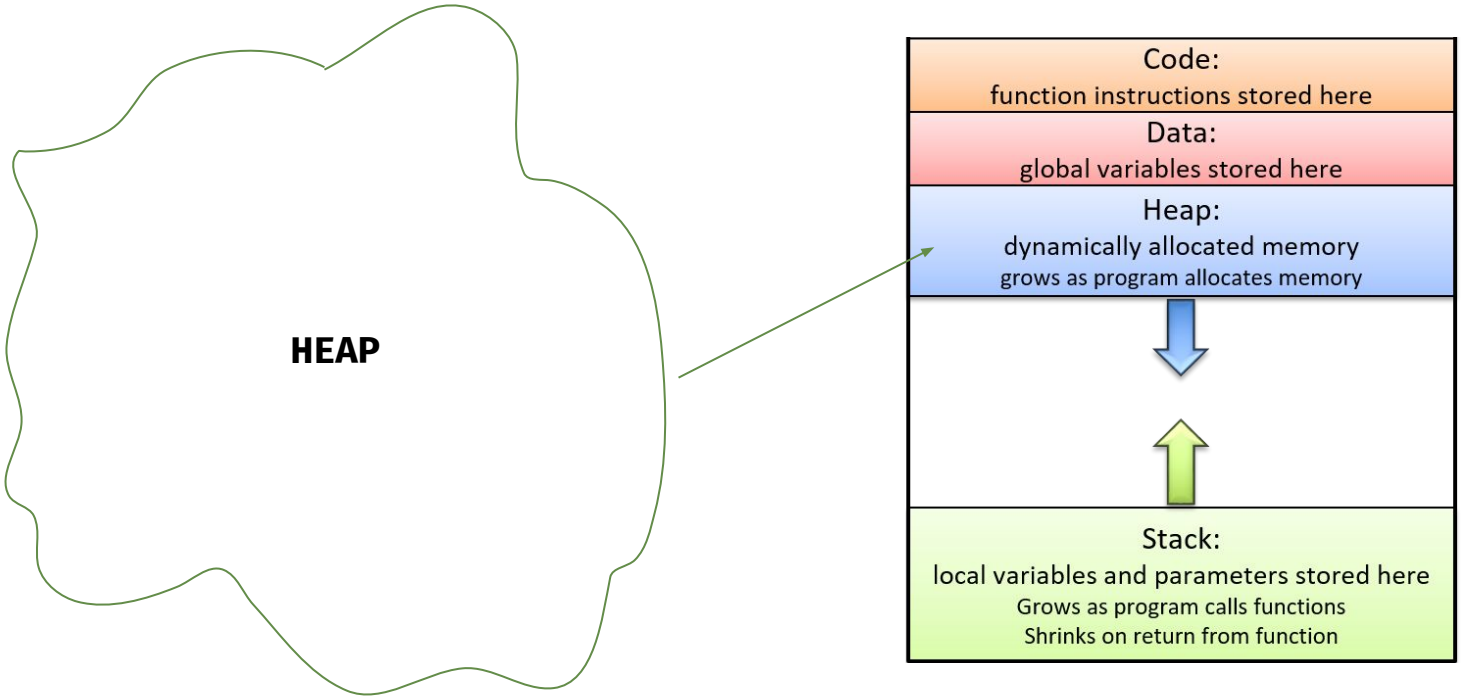
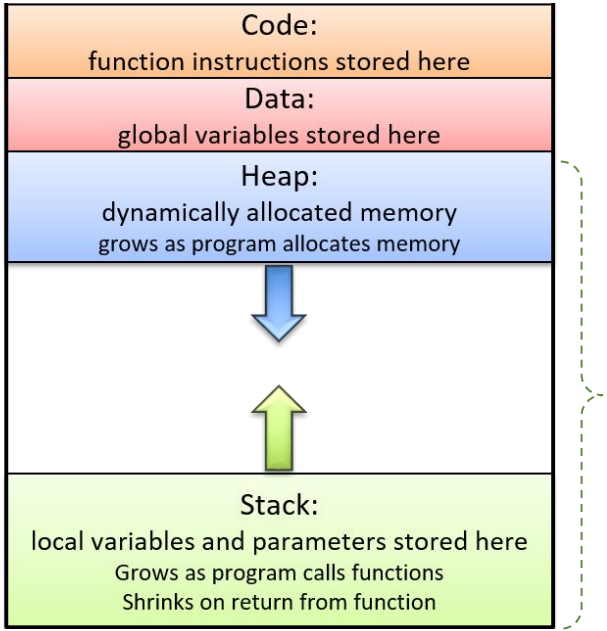
0X BARE C8 METAL

CMPSC 0XC8 ALLEGHENY COLLEGE 0X7E7


HEAP
Code:
function instructions stored here
Data:
global variables stored here
Heap:
dynamically allocated memory
grows as program allocates memory
Stack:
local variables and parameters stored here
Grows as program calls functions
Shrinks on return from function



Stack and Heap compete for space depending on program complexity and dynamic memory allocation during program execution.

After the Code and Data are allocated, whatever's left is Stack + Heap!

Pass by value

```
void teleport(int x, int y) {
    int tmp = x;
    x = y;
    y = tmp;
}
```

Scoped “locally”

x	2
y	1
tmp	1

```
int main(void) {
    …
}
```

```
int a = 1;
int b = 2;
```

a	1
b	2

What are **a** and **b** after all the operations?

Pass by value

```
void teleport(int *x, int *y) {
    int tmp = *x;
    *x = *y;
    *y = tmp;
}
```

x	2
y	1
tmp	1

```
int main(void) {
    int a = 1;
    int b = 2;
    …
}
```

a	2
b	1

What are **a** and **b** after all the operations?

Pass by value

C is a *pass by value* language.

```
void teleport(int *x, int *y)
{
    int tmp = *x;
    *x = *y;
    *y = tmp;
}

int main(void) {
    int a = 1;
    int b = 2;
    teleport(&a, &b);
    …
}
```

Pass by value

C is a *pass by value* language.

```
void teleport(int x, int y) {
    int tmp = x;
    x = y;
    y = tmp;
}
```

Creates *copies of variable values.*

```
int main(void) {
    int a = 1;
    int b = 2;
    teleport(a, b);
    …
}
```

Pass by value

C is a *pass by value* langu

```
void teleport(int *x, int *y) {
    int tmp = *x;
    *x = *y;
    *y = tmp;
}

int main(void) {
    int a = 1;
    int b = 2;
    teleport(a, b);
    …
}
```

Creates *copies of variable values.*


NYPD
int *a;
int &a;
imgflip.com

Pointers

```
void teleport(int *x, int *y)
{
    int tmp = *x;
    *x = *y;
    *y = tmp;
}

int main(void) {
    int a = 1;
    int b = 2;
    teleport(&a, &b);
    …
}
```

"Dereference": resolve the memory location and get the value.

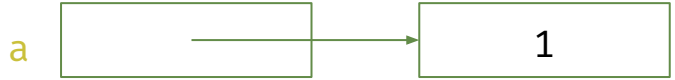
"Reference to": the memory location where the value in question lives.

Bonus question: can we guess *where* in memory these references live?

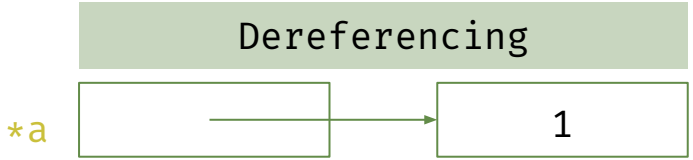
We'll answer later.

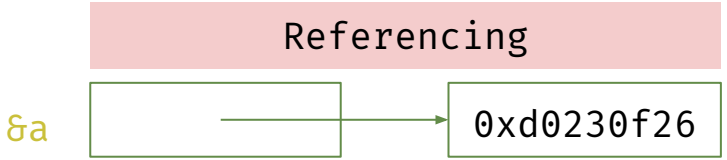
3 rules of pointers

1. Pointers always need to have a *pointee*.



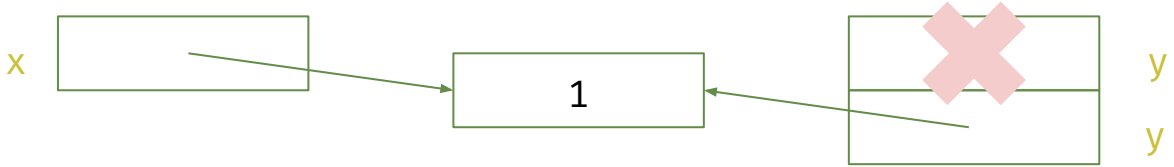
2. * ("Dereferencing") always yields a pointer *value*.





3. Pointer assignment between any two pointers makes them reference the same pointee.

```
int *y;
*x = *y;
```



Why pointers?

✅ 1. Share data between any two sections of a program easily
2. Create “ad hoc” *dynamic* data structures

A heap of trouble

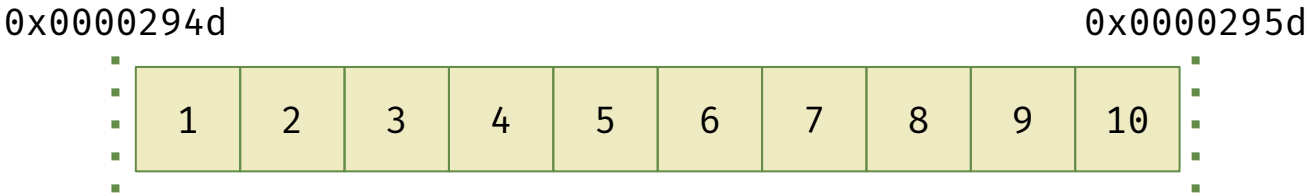
malloc	free
Allocates required amount of memory directly on the Heap.	Cleans up any Heap memory locations directly allocated using malloc.

A heap of trouble

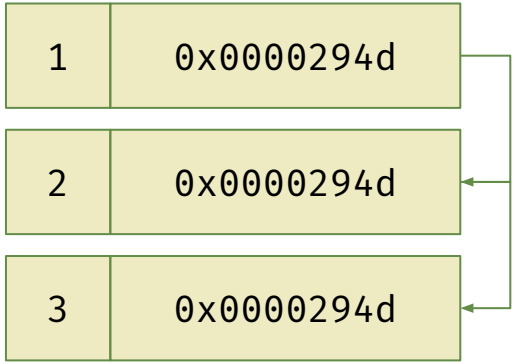
malloc	free
Allocates required amount of memory directly on the Heap.	Cleans up any Heap memory locations directly allocated using malloc.

Linking up

```
int numbers[10];
```




```
struct node {
    int id;
    struct node *next;
};
```

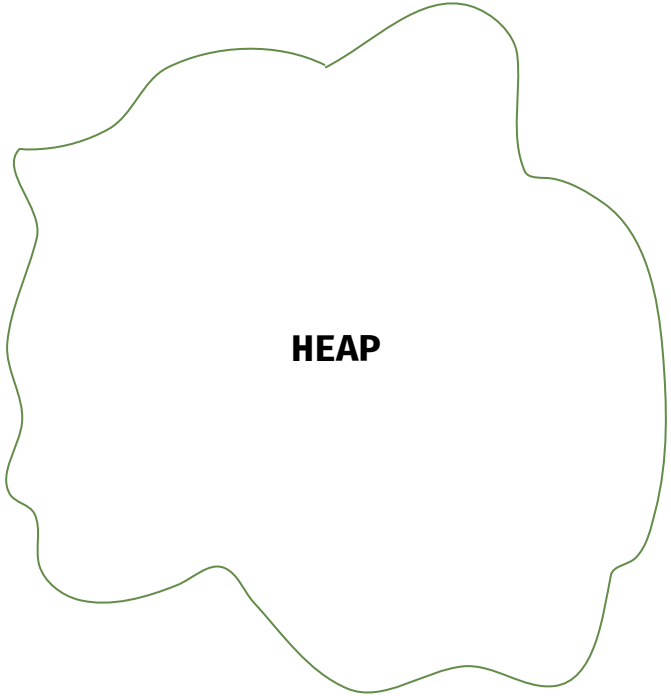


Linking up

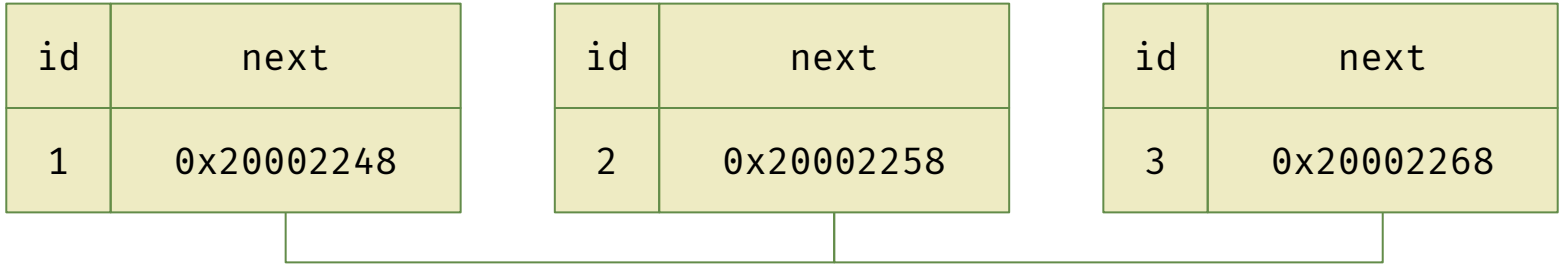
```
struct node {
    int id;
    struct node *next;
};
```

```
malloc(sizeof(node));
```

What size?



Linking up

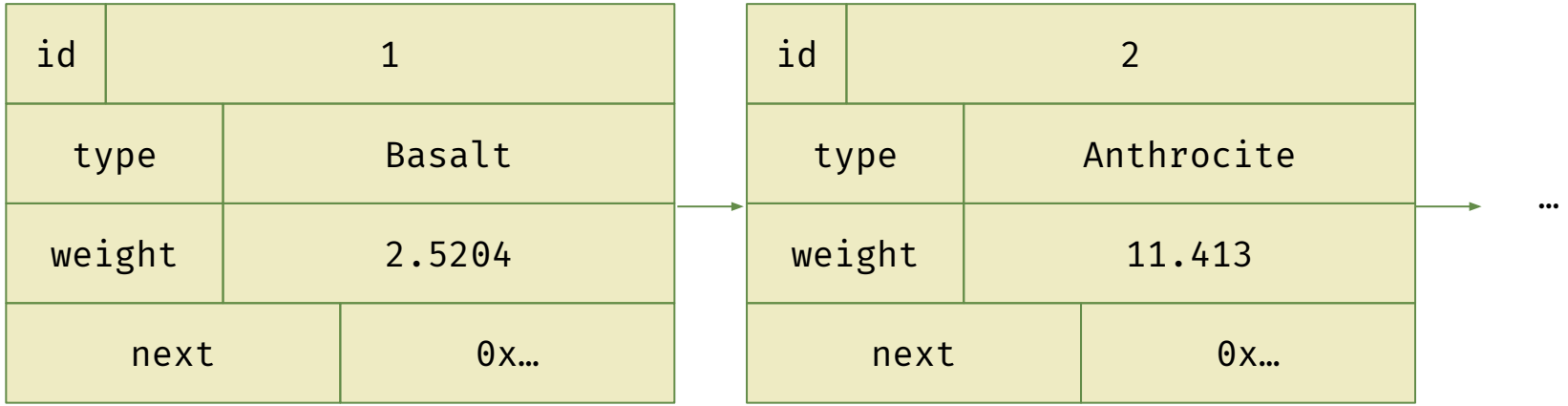


Each offset by 10_{16} → 16 bytes

Why pointers?

✅ 1. Share data between any two sections of a program easily
✅ 2. Create “ad hoc” *dynamic* data structures

Linking up


id
1
type
Basalt
weight
2.5204
next
0x…
id
2
type
Anthrocite
weight
11.413
next
0x…
…