From last time...

Arrays!

```
int arr[6] = {1, 15, 11, 0, 3, 7};
```

(0 1 2 3 4 5)

arr[3] →

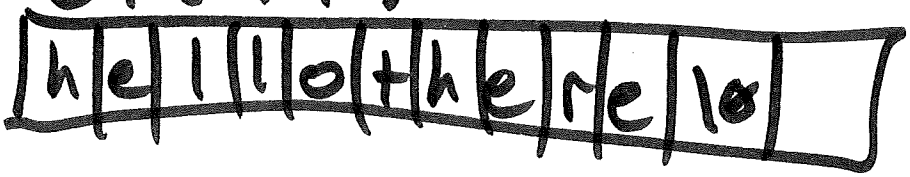
↗

dereference array @ 3rd elem.

arr[000] → *(arr + 1000)

Strings are just arrays of char

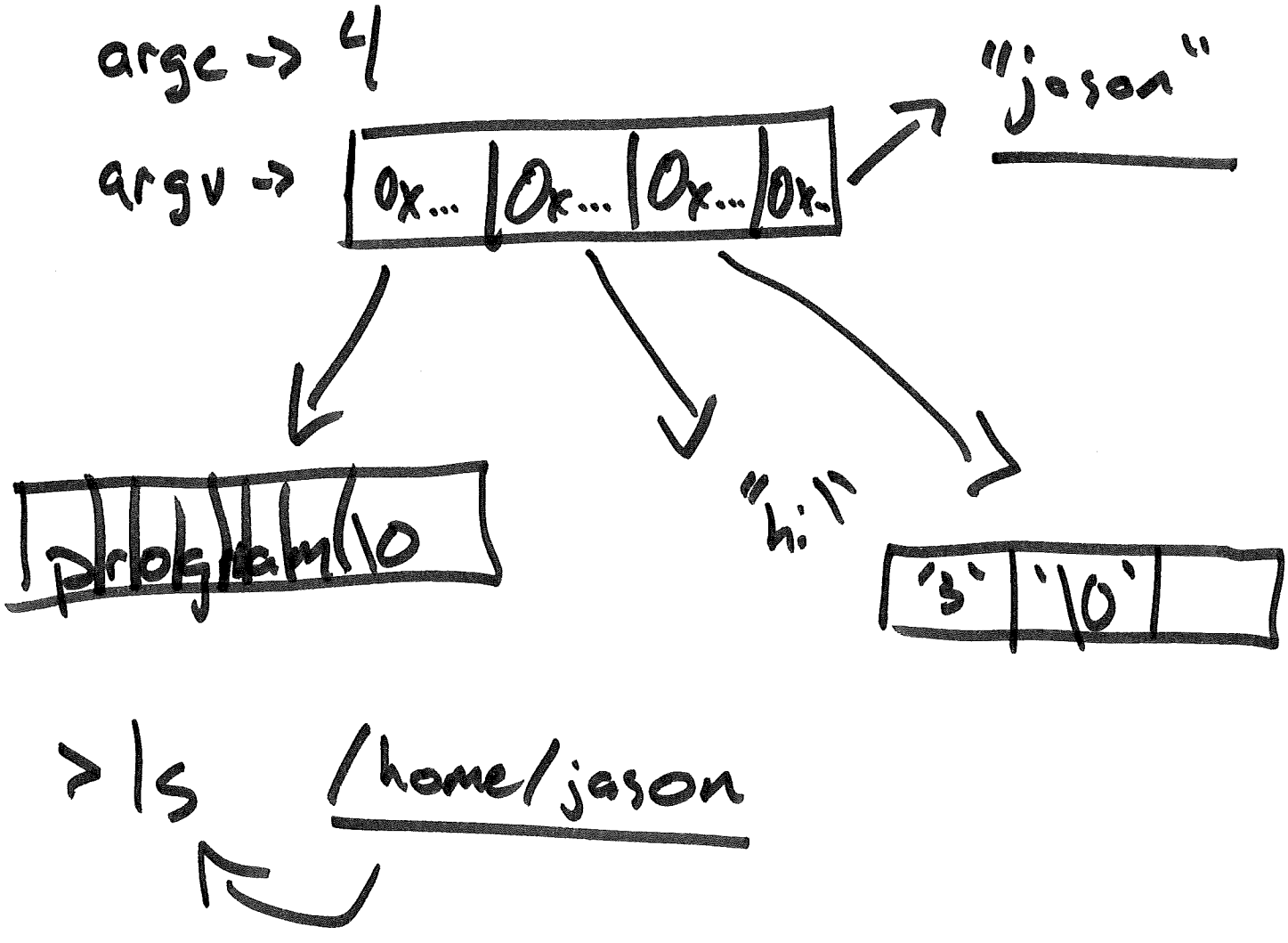
```
char string[10] = "hellothere";
```



string[5] → 't'

main(int argc, char *argv[])

> ./program hi 3 jason



> ls /home/jason

Pointer addition:

```
int *a = &i;      a -> 0x1000
short *b = &s;    b -> 0x2000
```

a+1 → 0x1001
0x1002
0x1004

↓

points to int

sizeof(int) → 4

b+1 → 0x2001
0x2002
0x2004

sizeof(short) → 2

&(a[1]) → 0x1004

b[1]

↓

&(*(a+1))

---

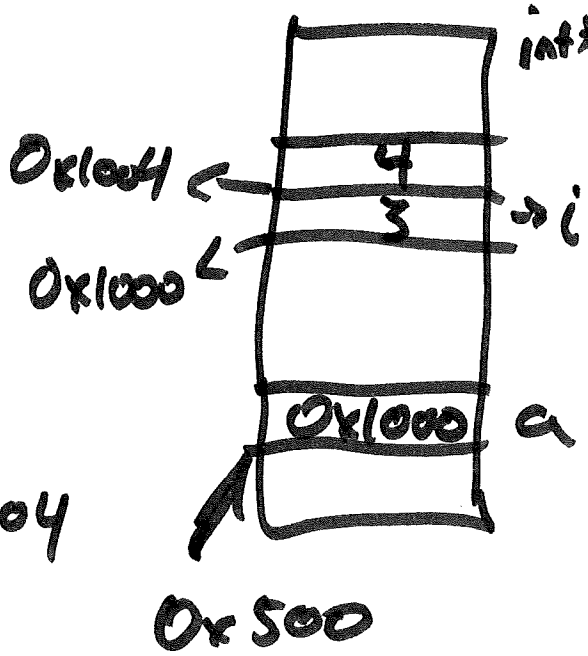
*(a+1) → 4

&(*(a+1)) →

0x1004

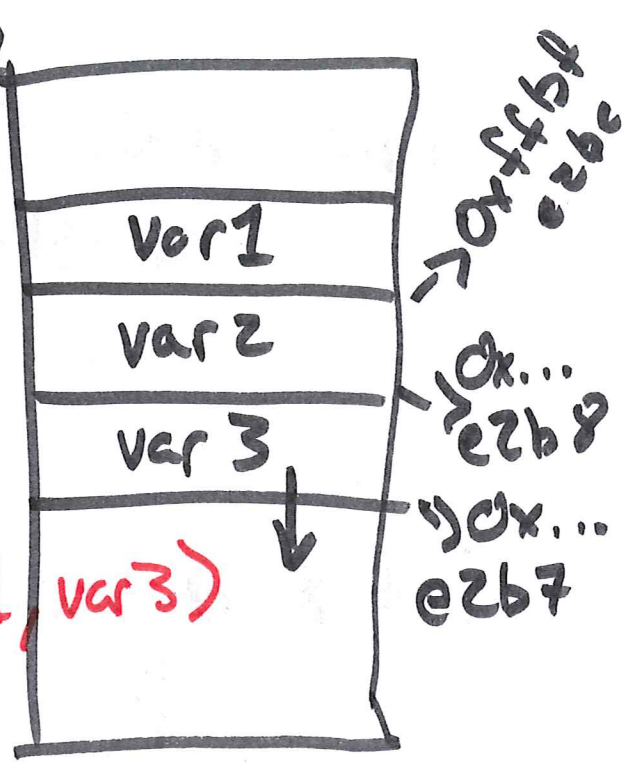
(a+1) → 0x1004

&a



```
int main(...) {
    int var1 = 1;
    int var2 = 2;
char int var3 = 3;
    return 0;
}
```

func1(var1, var3)

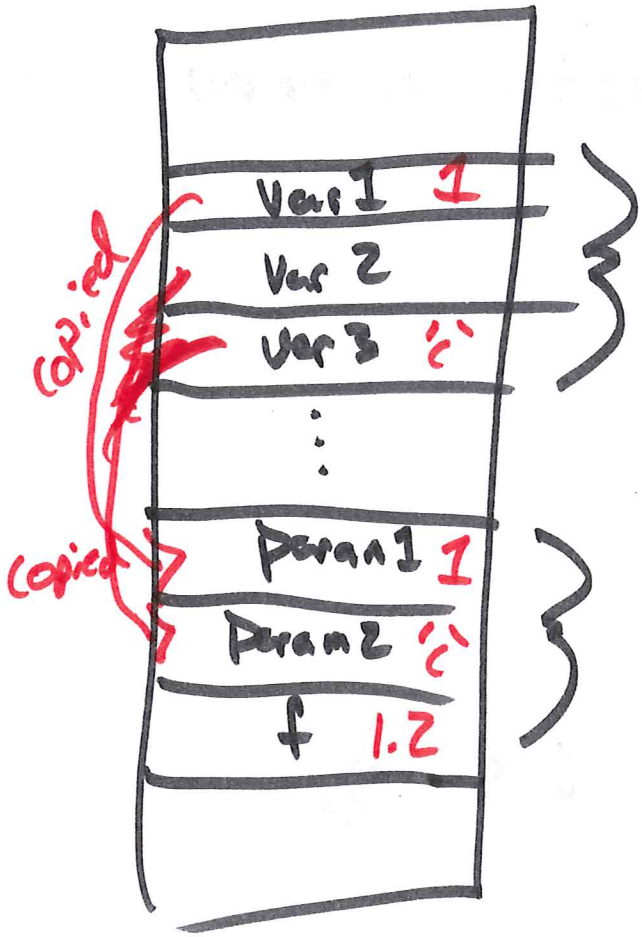


funtion local veriables live on Stack

Stack holds

- function local
- parameters

```
int func1 (int param1, char param2)
        float f = 1.2
        return 12;
}
```



int* a, b

int* @ int *a, *b

int a, b;

What if you don't know how much memory you need

→ char array[1000];

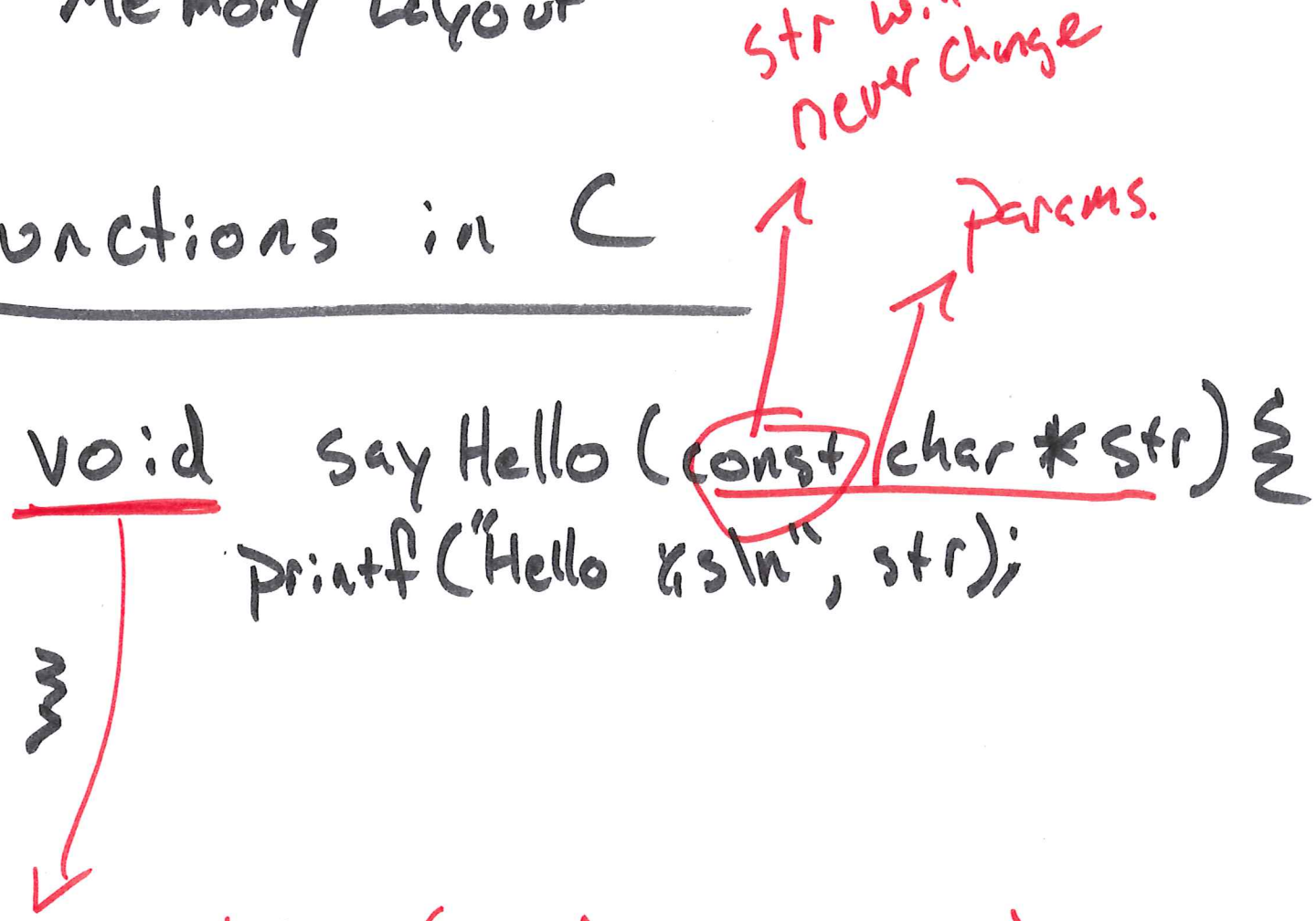
→ char *array;

```
while (1)
    char c = getc(stdin);
    if (c == EOF) {
        break;
    }
}
```

Where things actually live in memory

- Memory Layout

functions in C




```
int multiply (int a, int b) {
    return a*b;
}

void addOne (int *a) {
    *a = *a + 1;
}
```

implies ↳ pointer mutable