Lecture 6
Pointers and addresses

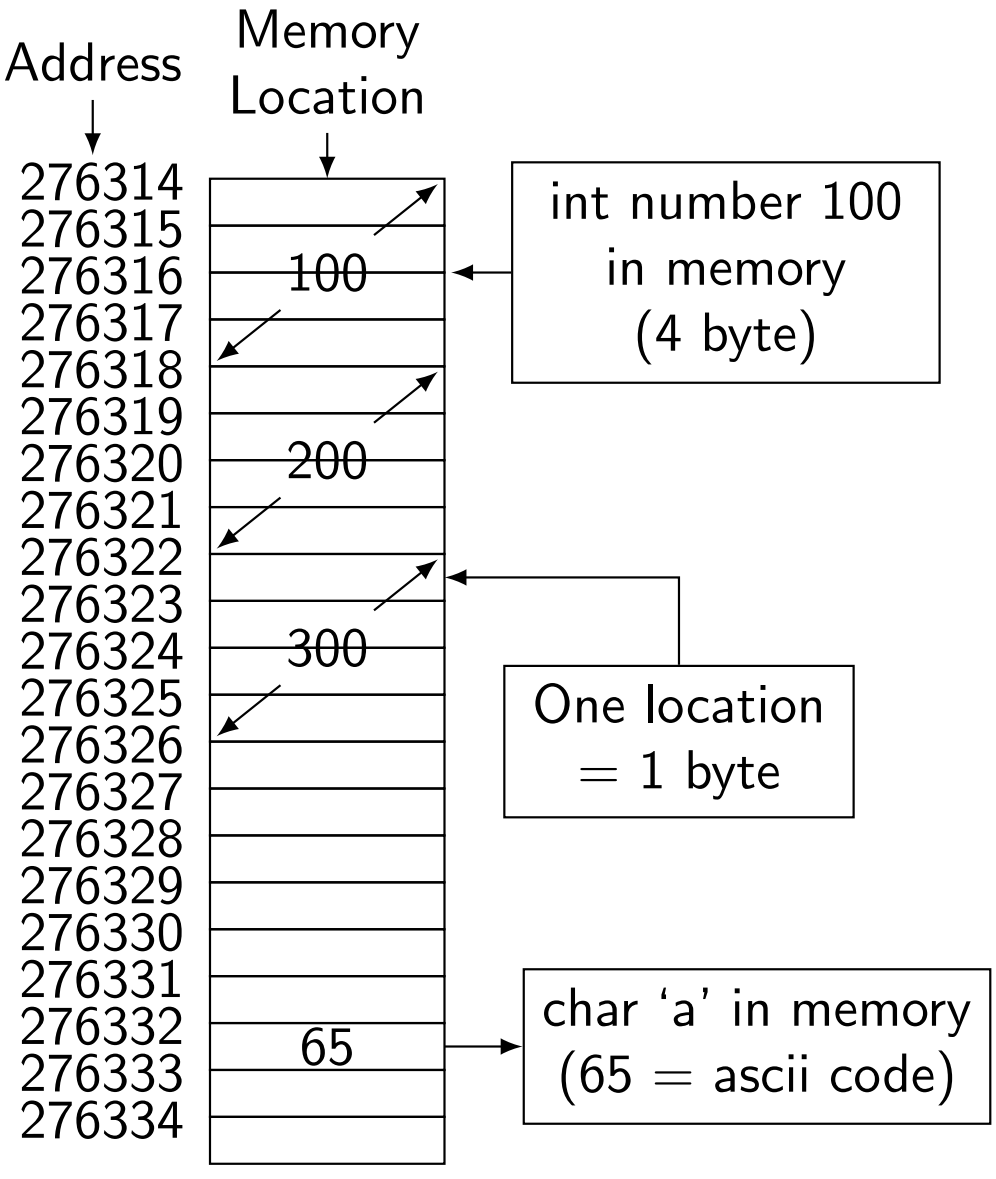
Test is coming

- Data types
- Functions
- I/O operations
- Branching (if, switch)
- Loops

Have a look at the example tests!

The memory

Is where variables live




```
...
int a=100;
int b=200;
int c=300;
char d='a';
...
```

- Memory is continous
- All variables are stored in memory
- ... and functions

New data types - pointers

declared with a *

- For every type there is a pointer to it
- Use *
- Pointers are used to store **addresses** of variables
- Reside in memory, as any other variable

```
int *pi;
float *pf;
double *pd;
char *pc;
void *pv;
```

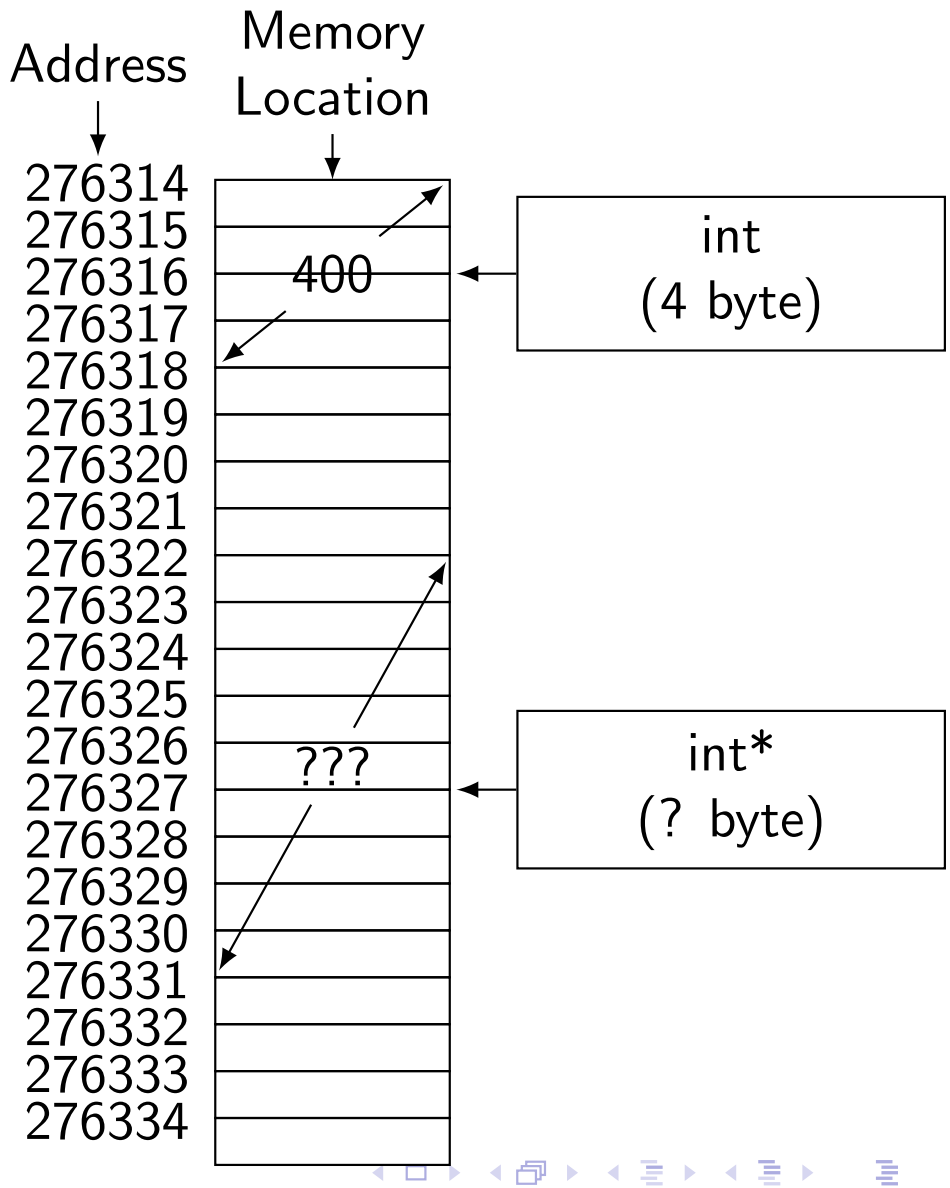
But also Pointer to pointer ...

```
int **ppi;
float ***pf;
...
void **pv;
```

Pointers
sizeof

- What is *sizeof(int*)*
- and *sizeof(double*)*
- Examples follow
- Depands on a system ...

```
int a=400;
int *p = 10; //p points to
     memory address 10, can
     we access it?
```

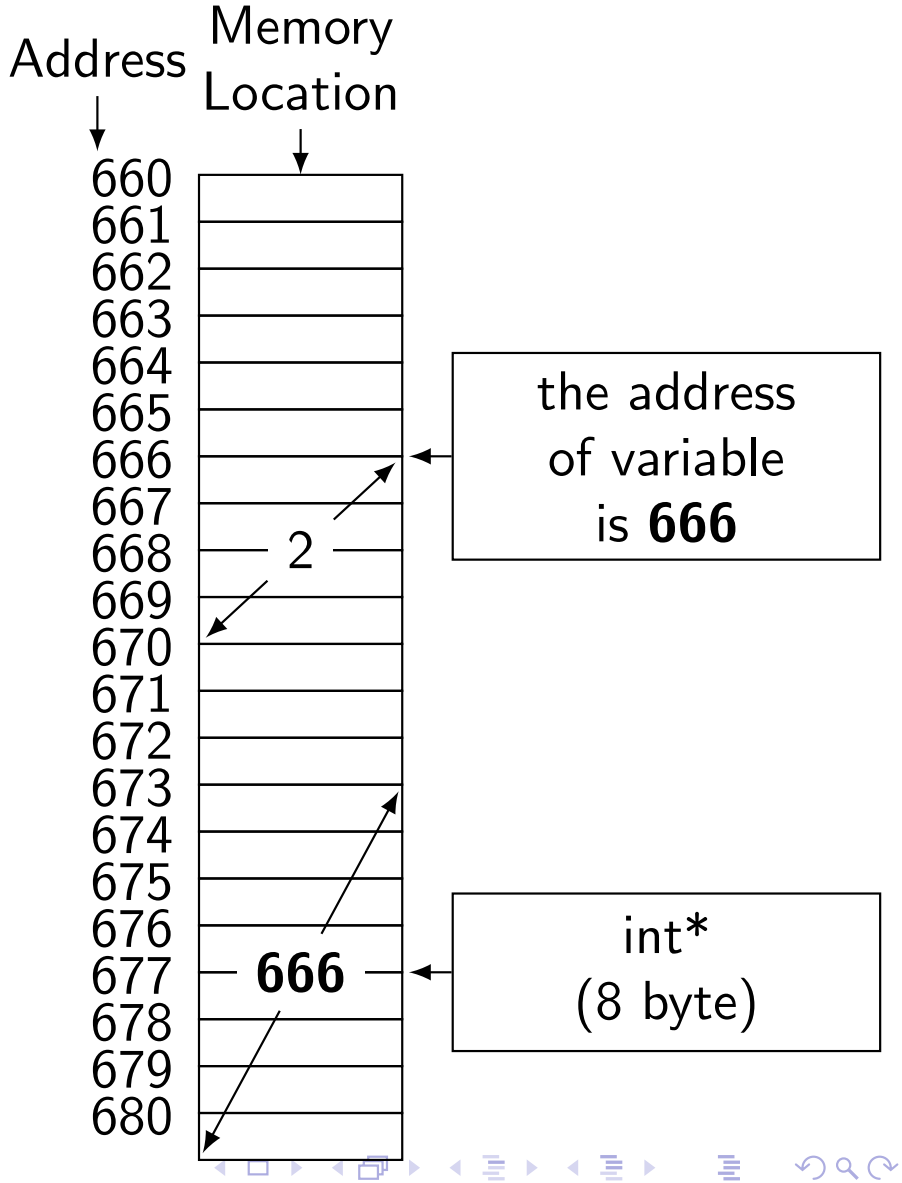


Retrieve the address

The & operator

- Remember the *scanf()*?
- & is used to retrieve an address of a variable in memory
- & returns the beginning of the space in memory where a variable is

```
int satan=2;//this is an evil int
int *p = &a; //p stores adress of s
```



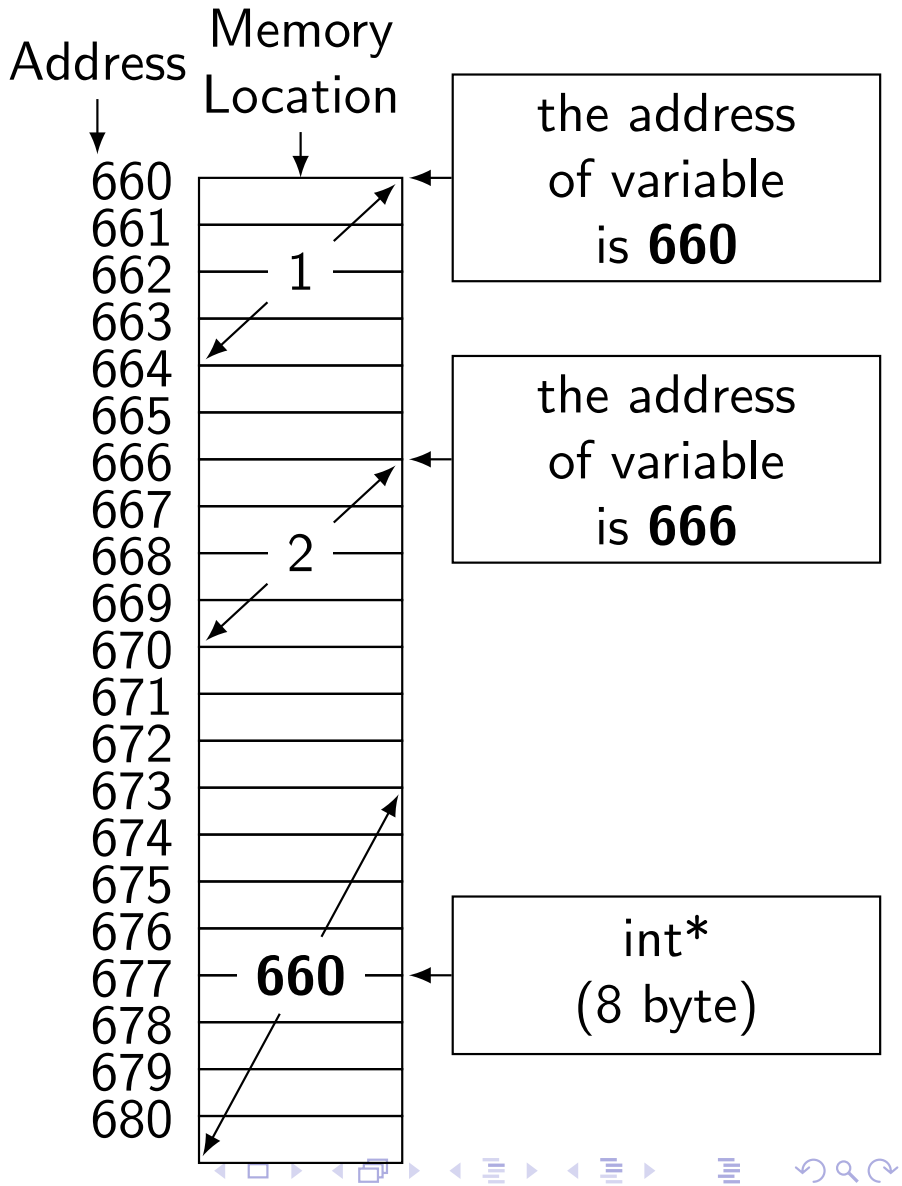
Retrieve the address

The & operator

- Remember the *scanf()*?
- & is used to retrieve an address of a variable in memory
- & returns the beginning of the space in memory where a variable is

```
int satan=2;//this is an evil int
int *p = &a; //p stores adress of s

int good=1;//this is a good int
p=&good;//p stores adress of good
```



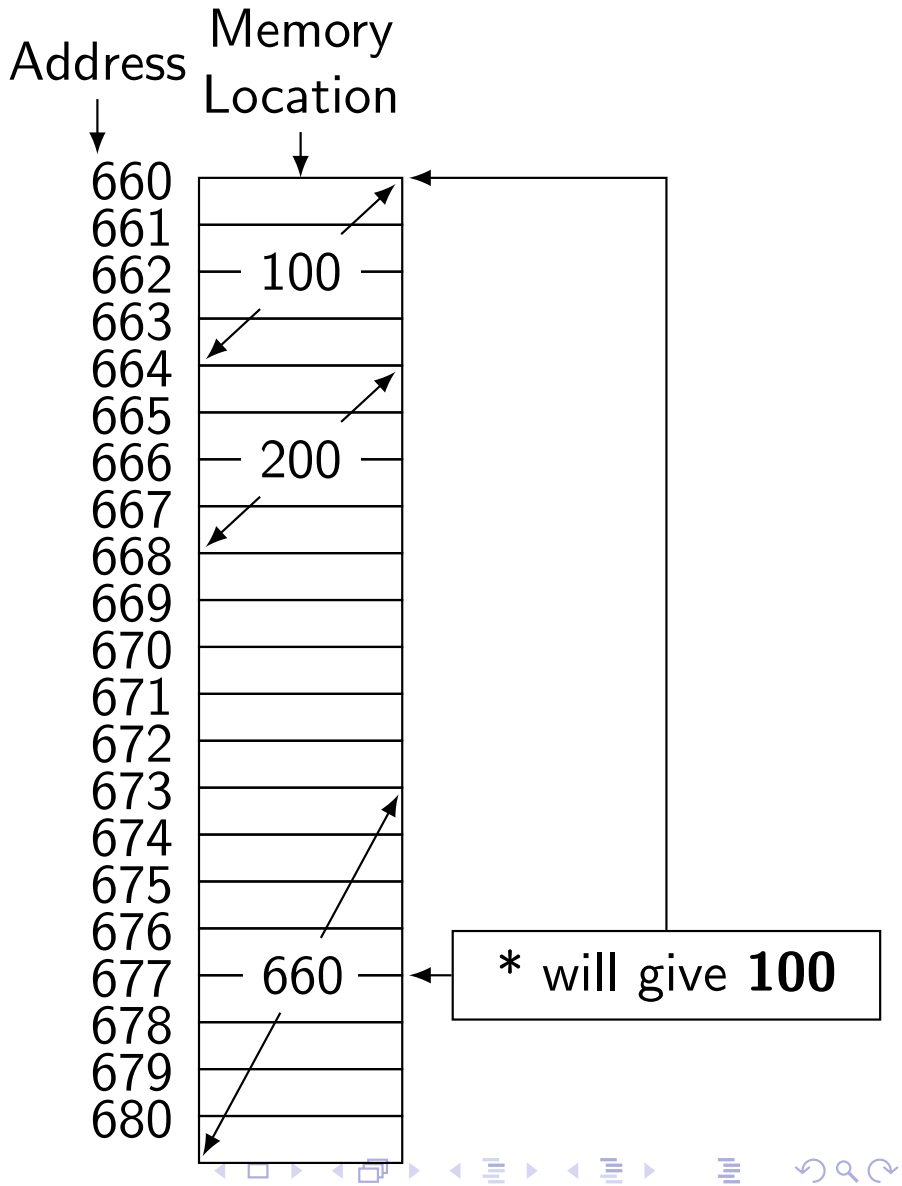
Retrive the variable

The * operator

- To get value, of variable, pointed by the pointer
- Use * operator on the pointer

```
int a=100;
int b=200;
int *p=&a;
printf("%d\n", *p);
```

So for *int* ** (pointer to pointer) the ** will give a value ...

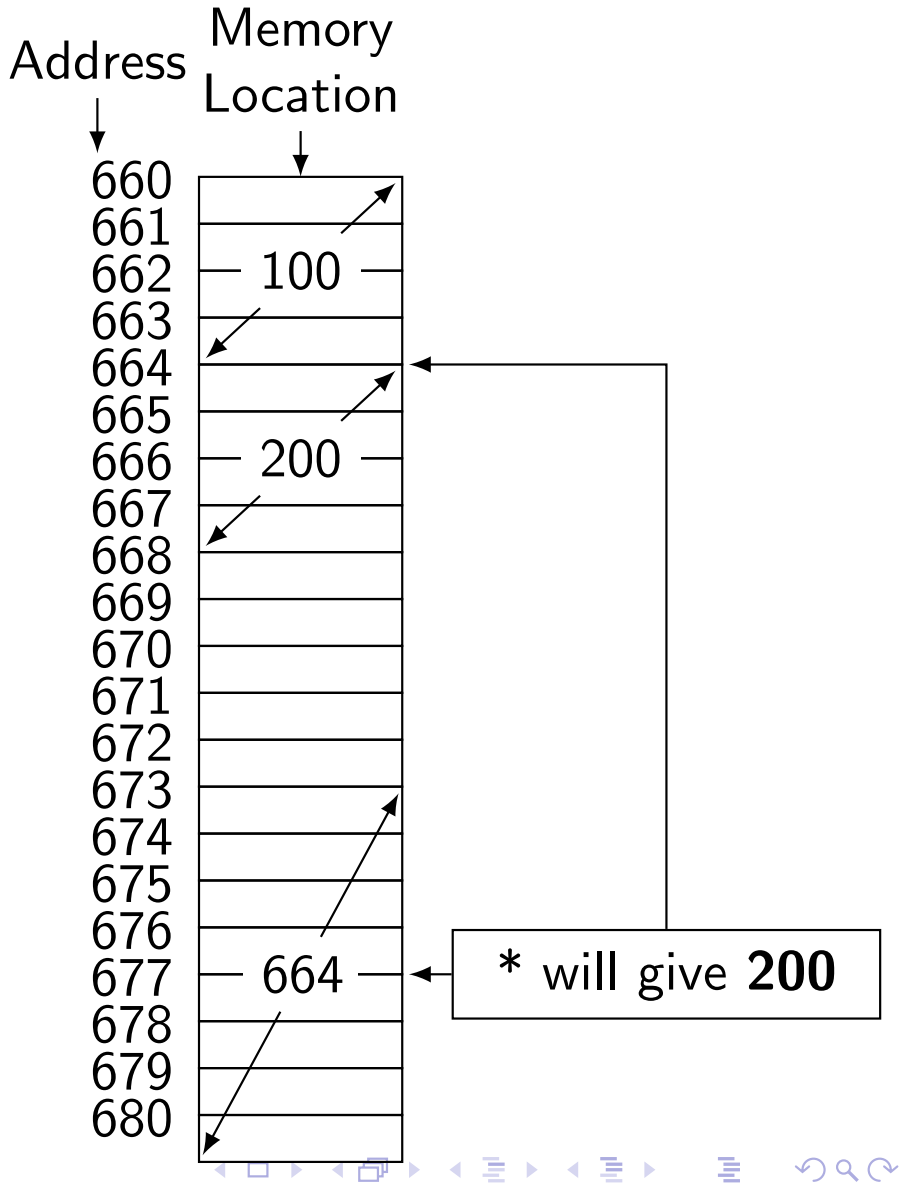


Retrieve the variable

The * operator

- To get value, of variable, pointed by the pointer
- Use * operator on the pointer

```
int a=100;
int b=200;
int *p=&a;
printf("%d\n", *p);
p=&b;
printf("%d\n", *p);
```

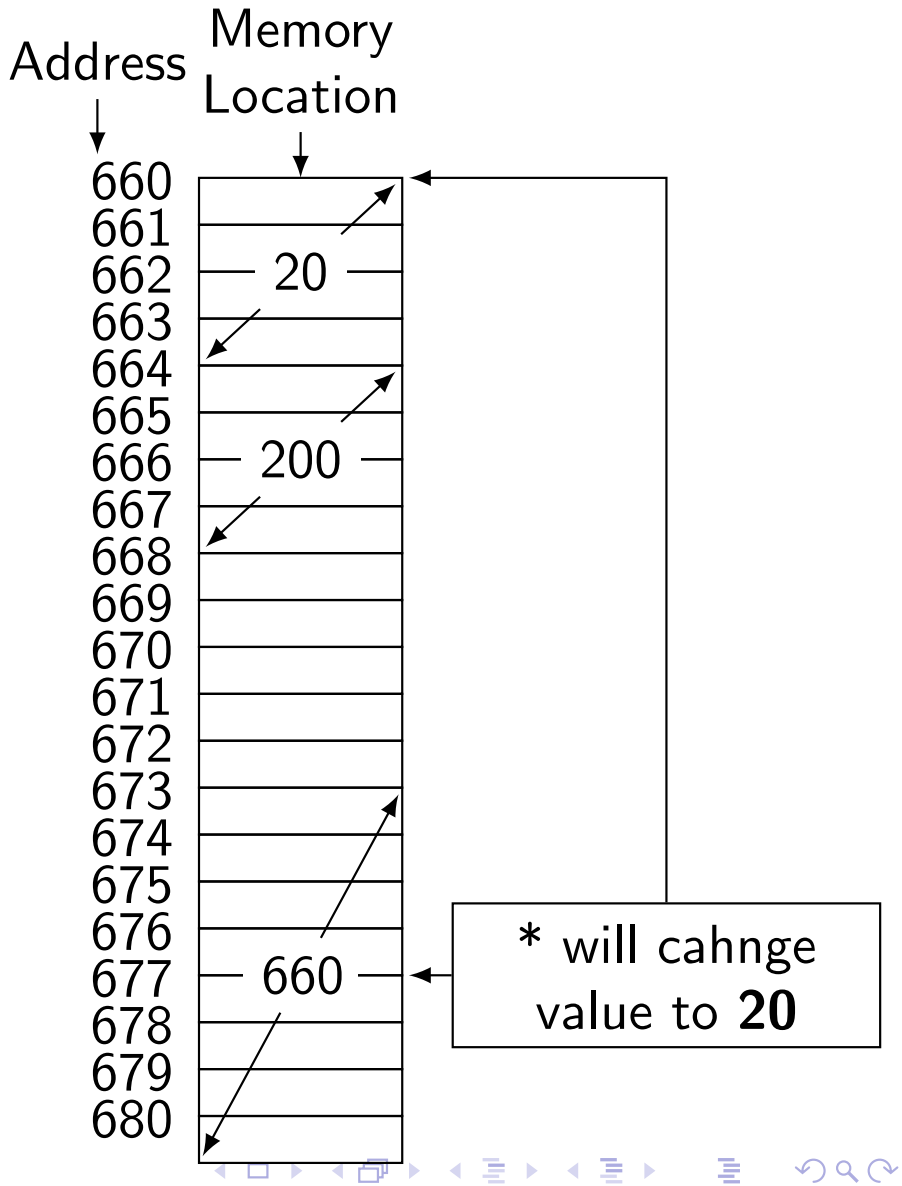


Change the variable

The * operator

- * can be used to change value pointed by the pointer
- Use * operator on the pointer and ...

```
int a=100;
int b=200;
int *p=&a;
printf("%d\n", *p);
*p = 20;
printf("%d\n", *p);
```

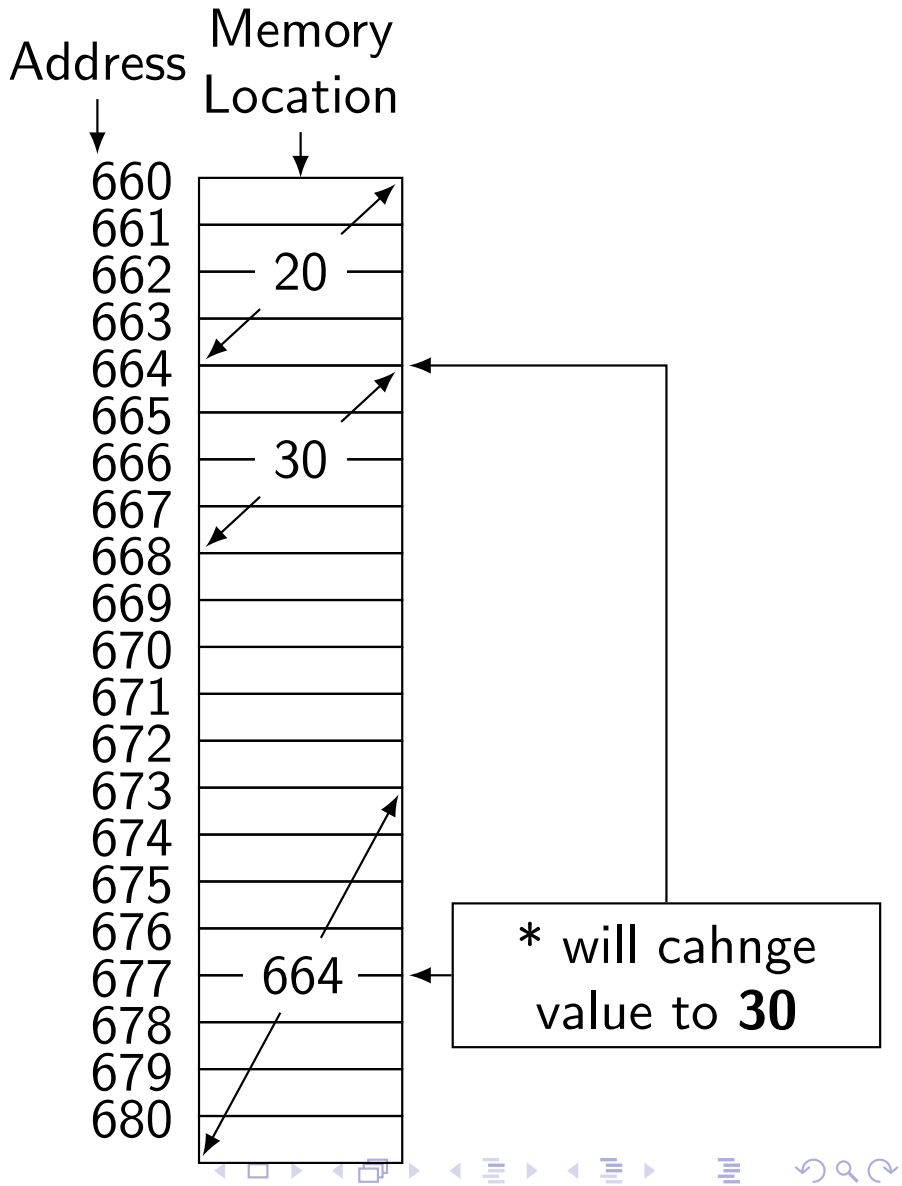


Change the variable

The * operator

- * can be used to change value pointed by the pointer
- Use * operator on the pointer and ...

```
int a=100;
int b=200;
int *p=&a;
printf("%d\n", *p);
*p = 20;
printf("%d\n", *p);
p=&b;
*p=30;
```



Printing the address stored by a pointer

%p ... or %d

```
int a=10;
printf("%p\n", &a);
int *p = &a;
printf("%p\n", p)

printf("%p\n", &p);??
```

Pointer arithmetic

+ -

```
int a=10;
printf("%p\n", &a);
int *p = &a;
printf("%p\n", p+1)?
```

How many *ints* I could hide in a single *double* ...
I should not ...

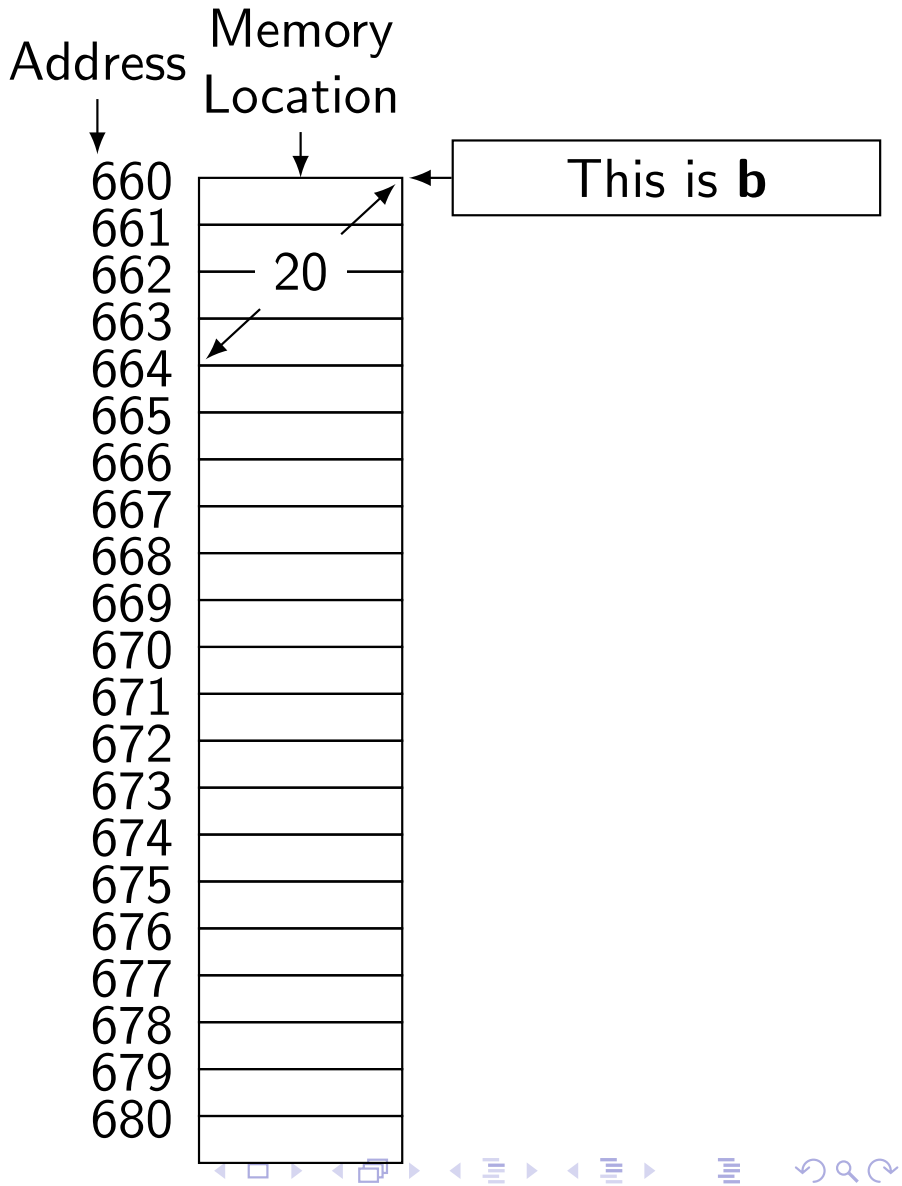
Function with arguments

Passed by value

- Only the value is send to a function

```
void fun(int a){
  a = 500;
}

int main(){
  inb b=20;
  fun(b);
  //b?
}
```



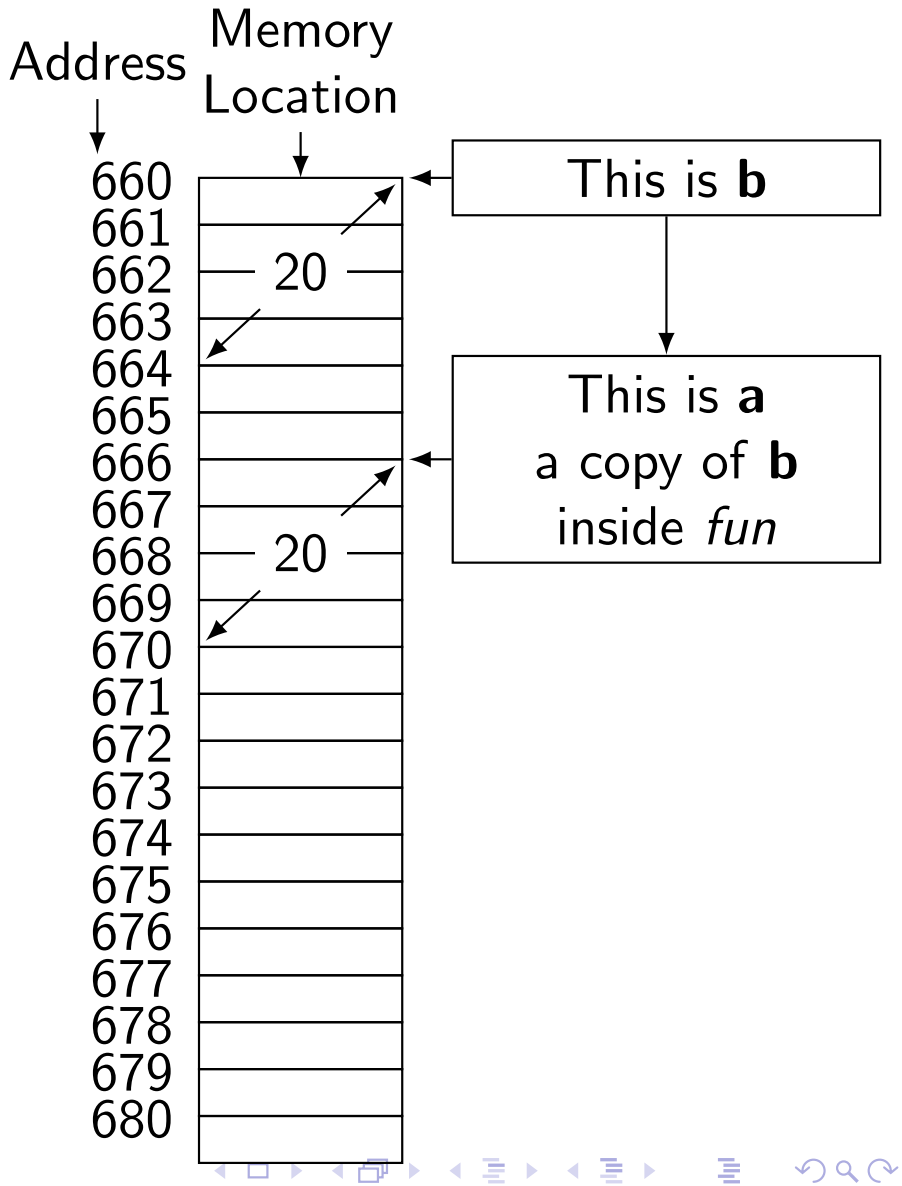
Function with arguments

Passed by value

- Only the value is send to a function

```
void fun(int a){
  a = 500;
}

int main(){
  inb b=20;
  fun(b);
  //b?
}
```



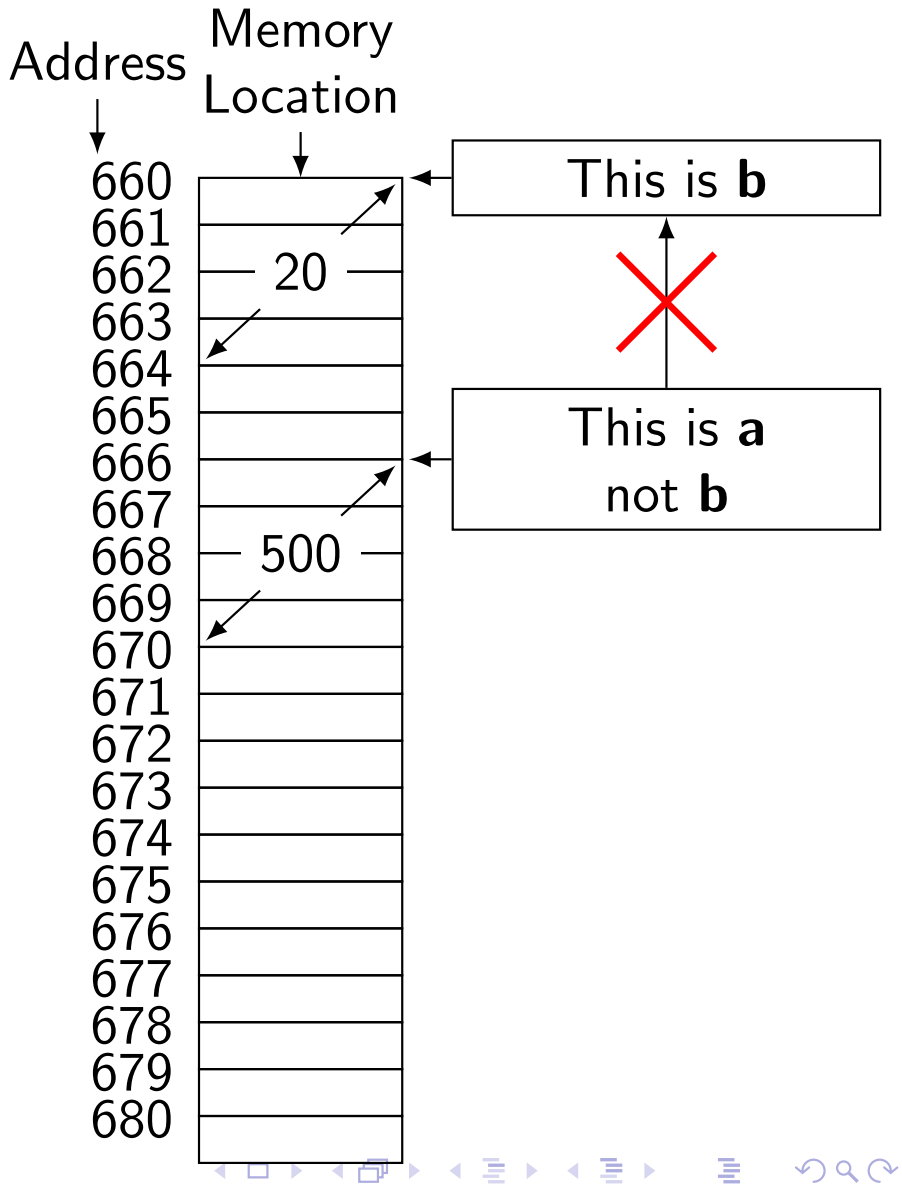
Function with arguments

Passed by value

- Only the value is send to a function

```
void fun(int a){
  a = 500;
}

int main(){
  inb b=20;
  fun(b);
  //b?
}
```



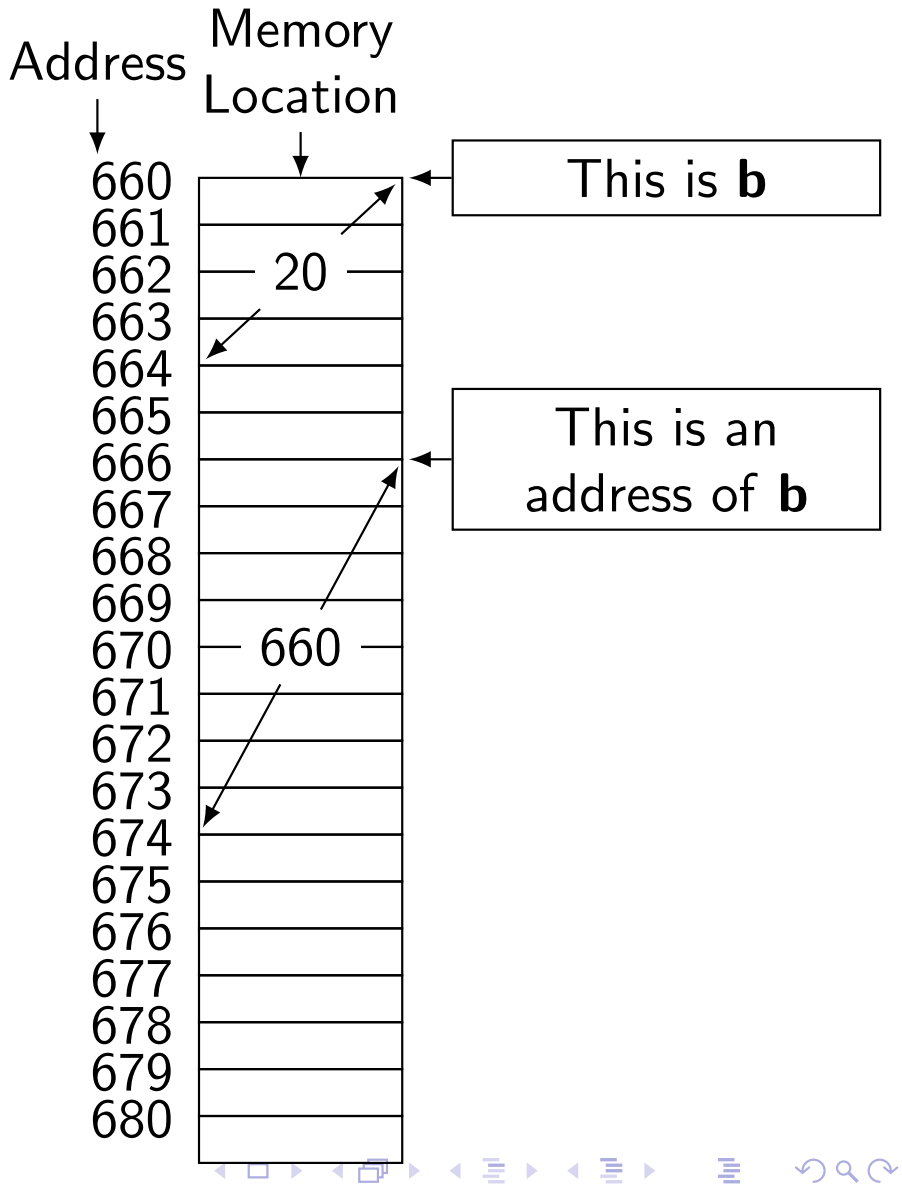
Function with arguments

Pass an address?

- What if we pass an address to a variable
- Than the function "knows" where the variable is stored
- The function works on tha variable
- ... not a copy

```
void fun(int* a){
  *a = 500;
}

int main(){
  inb b=20;
  fun(&b);//like scanf
  //b?
}
```



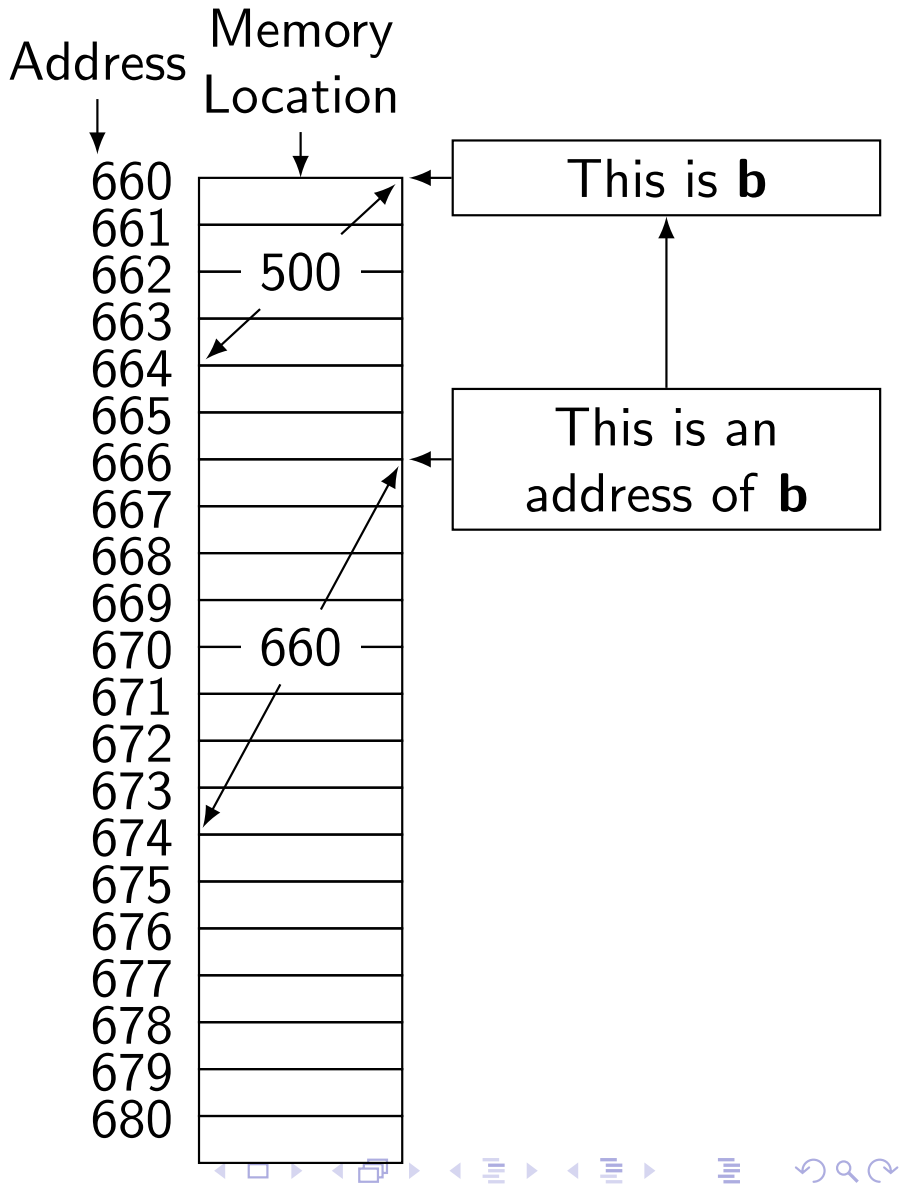
Function with arguments

Pass an address?

- What if we pass an address to a variable
- Than the function "knows" where the variable is stored
- The function works on tha variable
- ... not a copy

```
void fun(int* a){
  *a = 500;
}

int main(){
  inb b=20;
  fun(&b);//like scanf
  //b?
}
```



Temporary page!

LaTeX was unable to guess the total number of pages correctly. As the
was some unprocessed data that should have been added to the final
page this extra page has been added to receive it.
If you rerun the document (without altering it) this surplus page will g
away, because LaTeX now knows how many pages to expect for this
document.