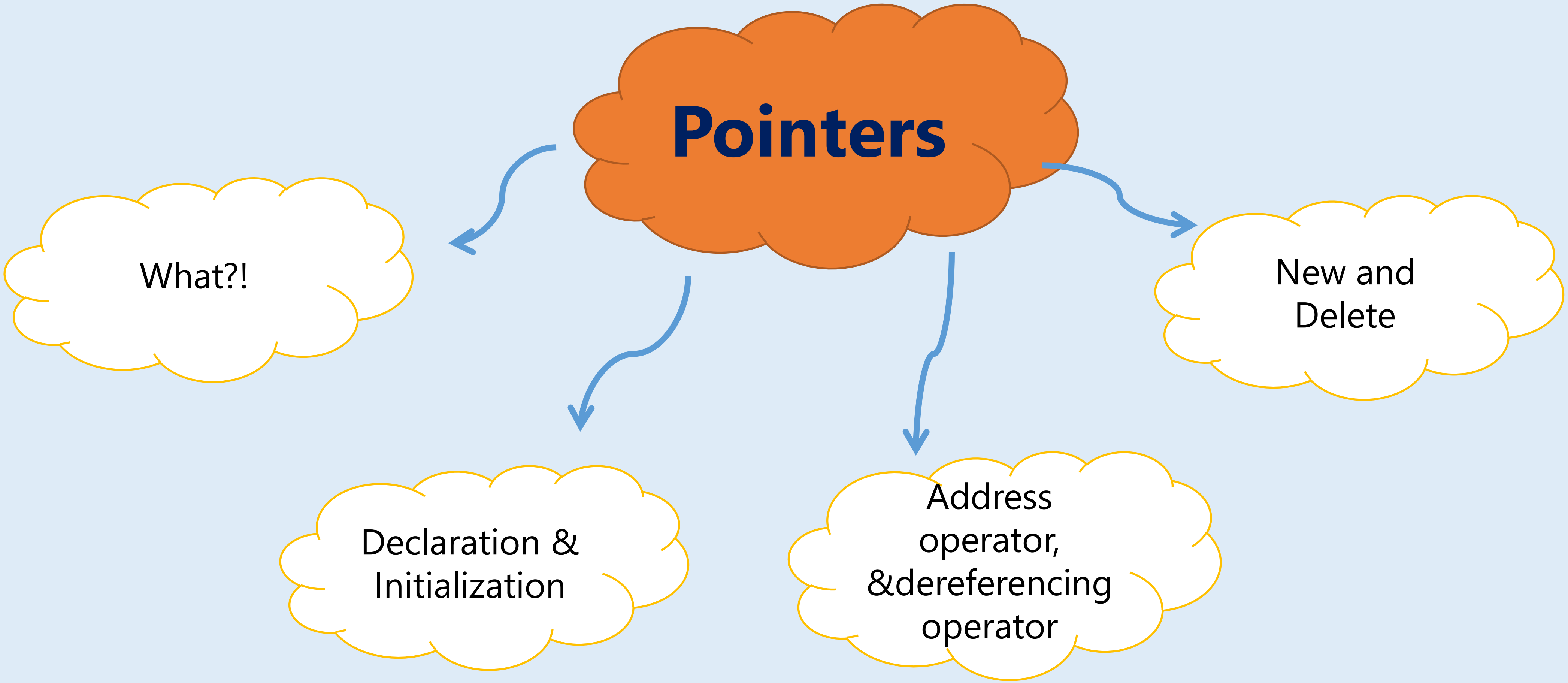
Lab #9

Pointers

Structured Programming 2017/2018

Today’s Lab



Pointers

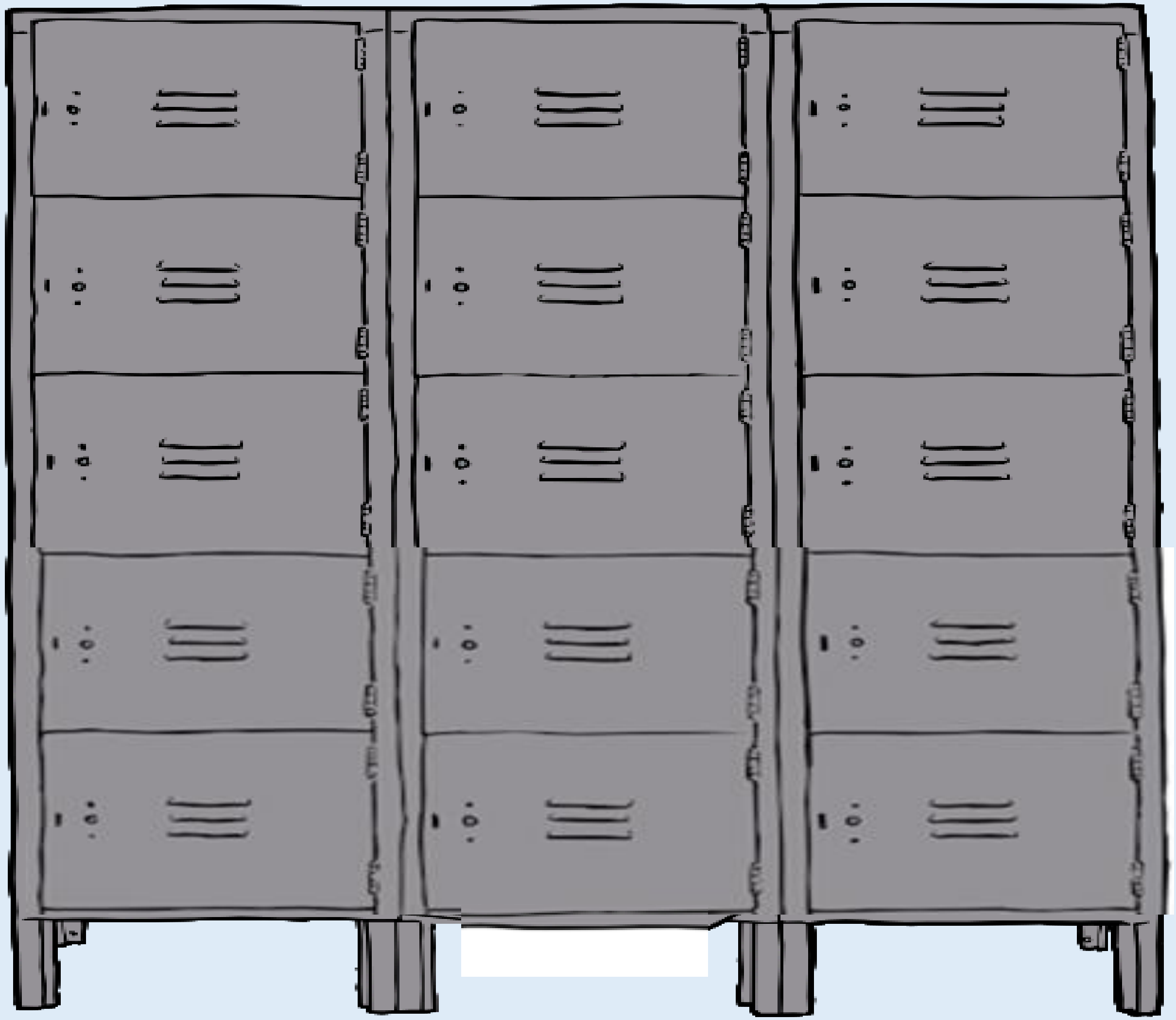
What?!

Declaration & Initialization

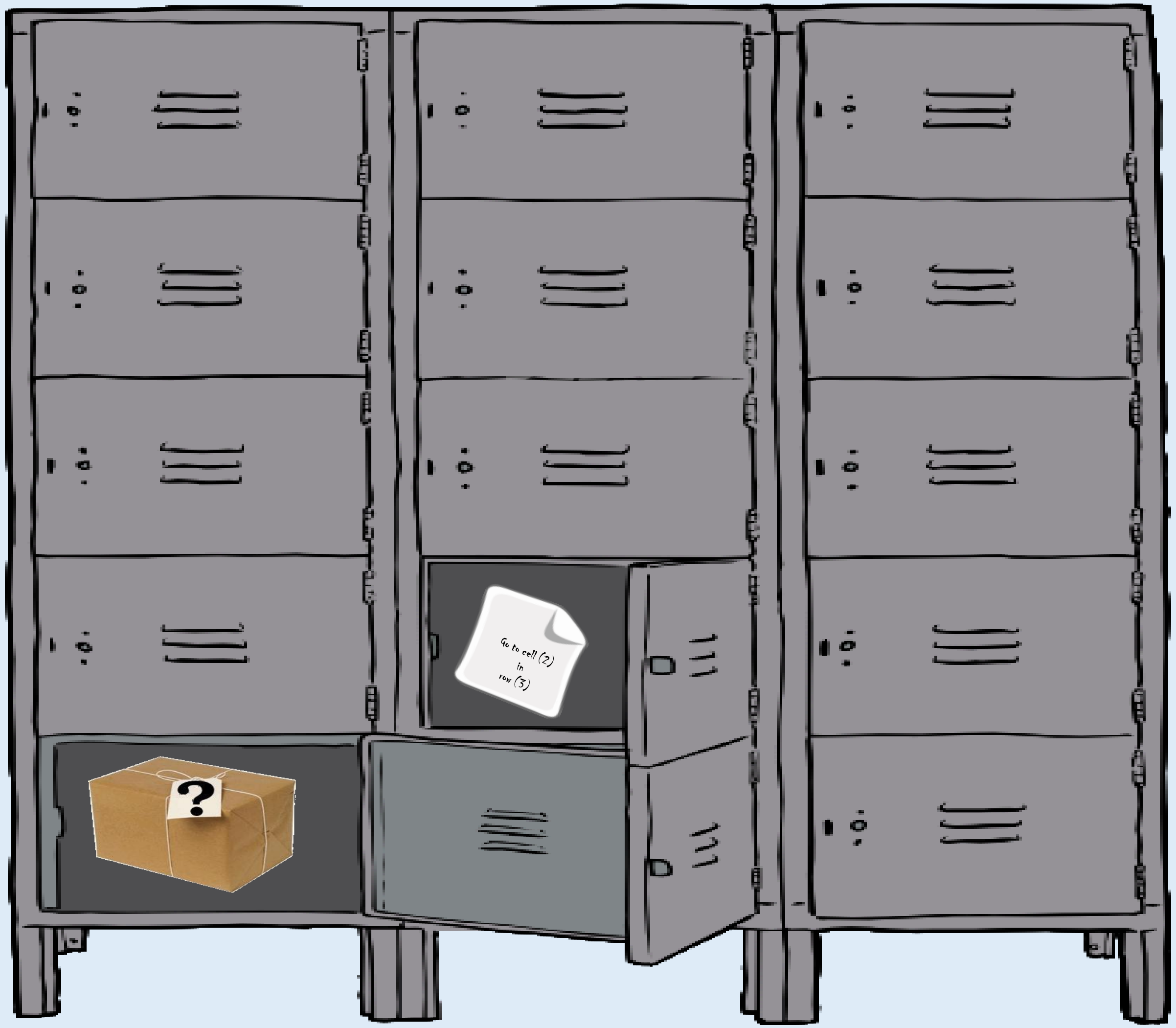
Address operator, &dereferencing operator

New and Delete

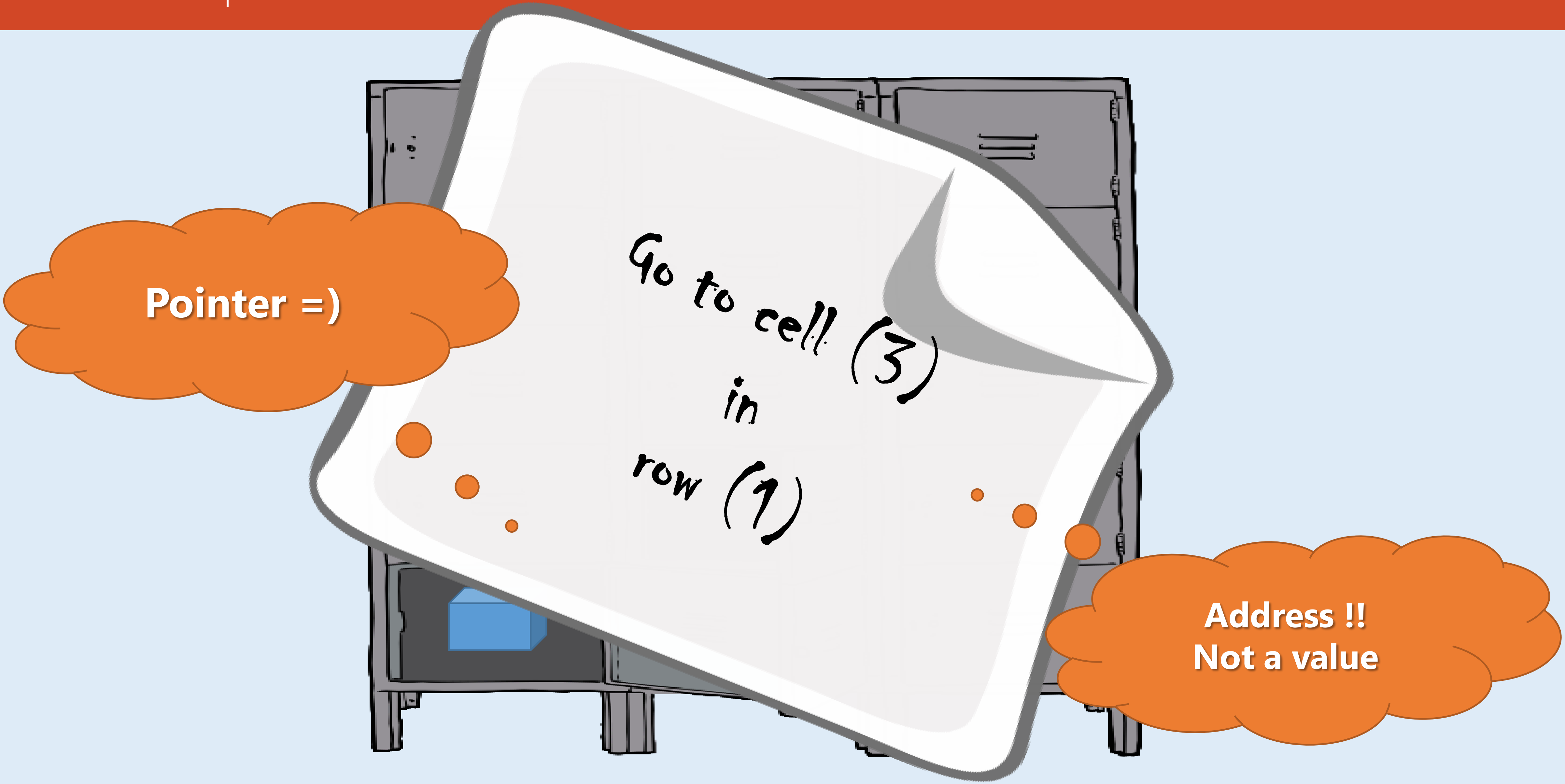
What is a pointer?



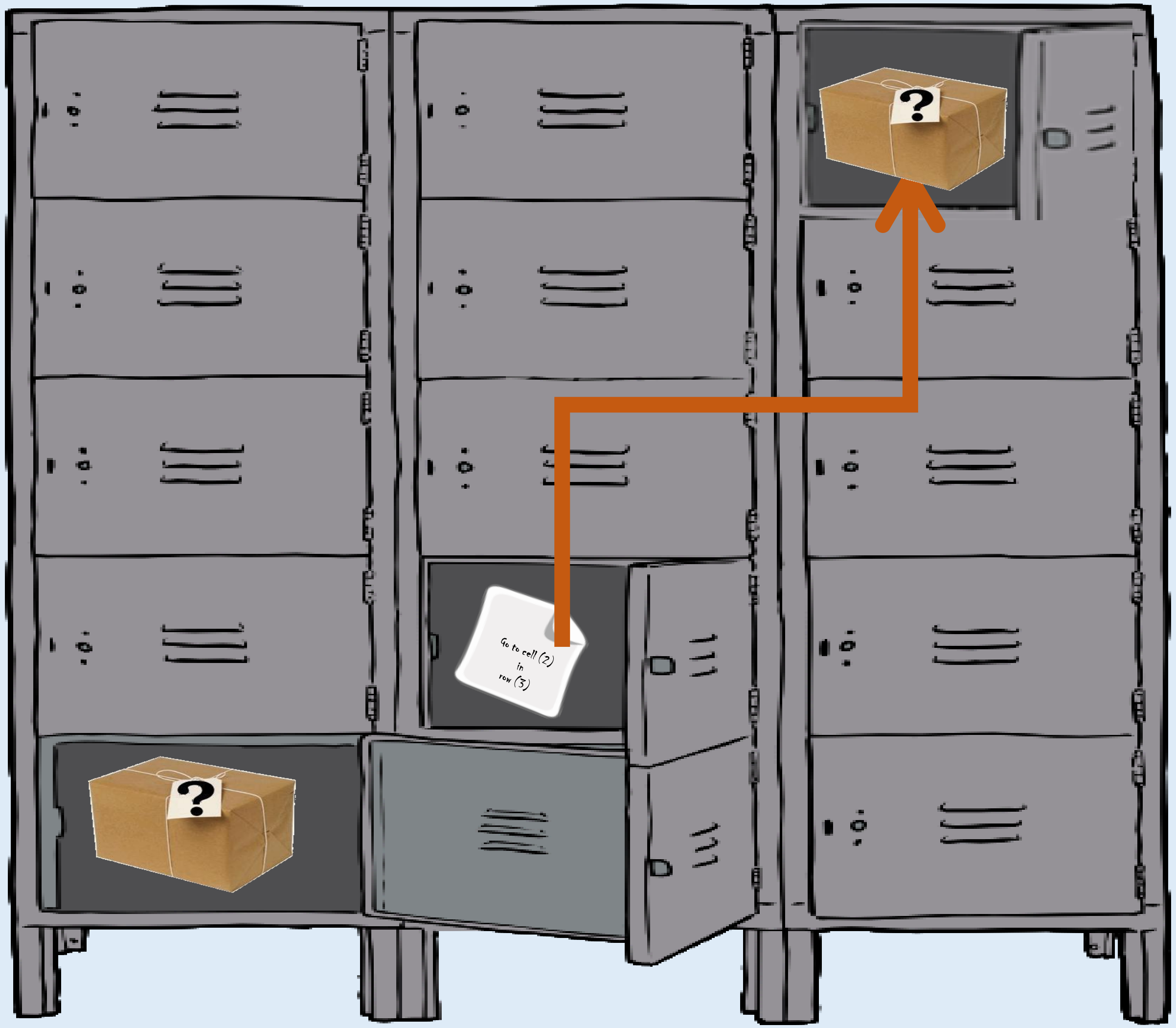
What is a pointer?



What is a pointer?

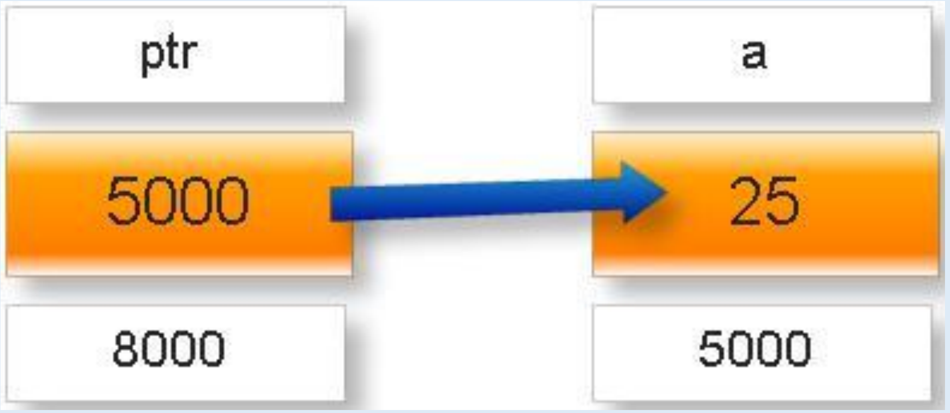


What is a pointer?

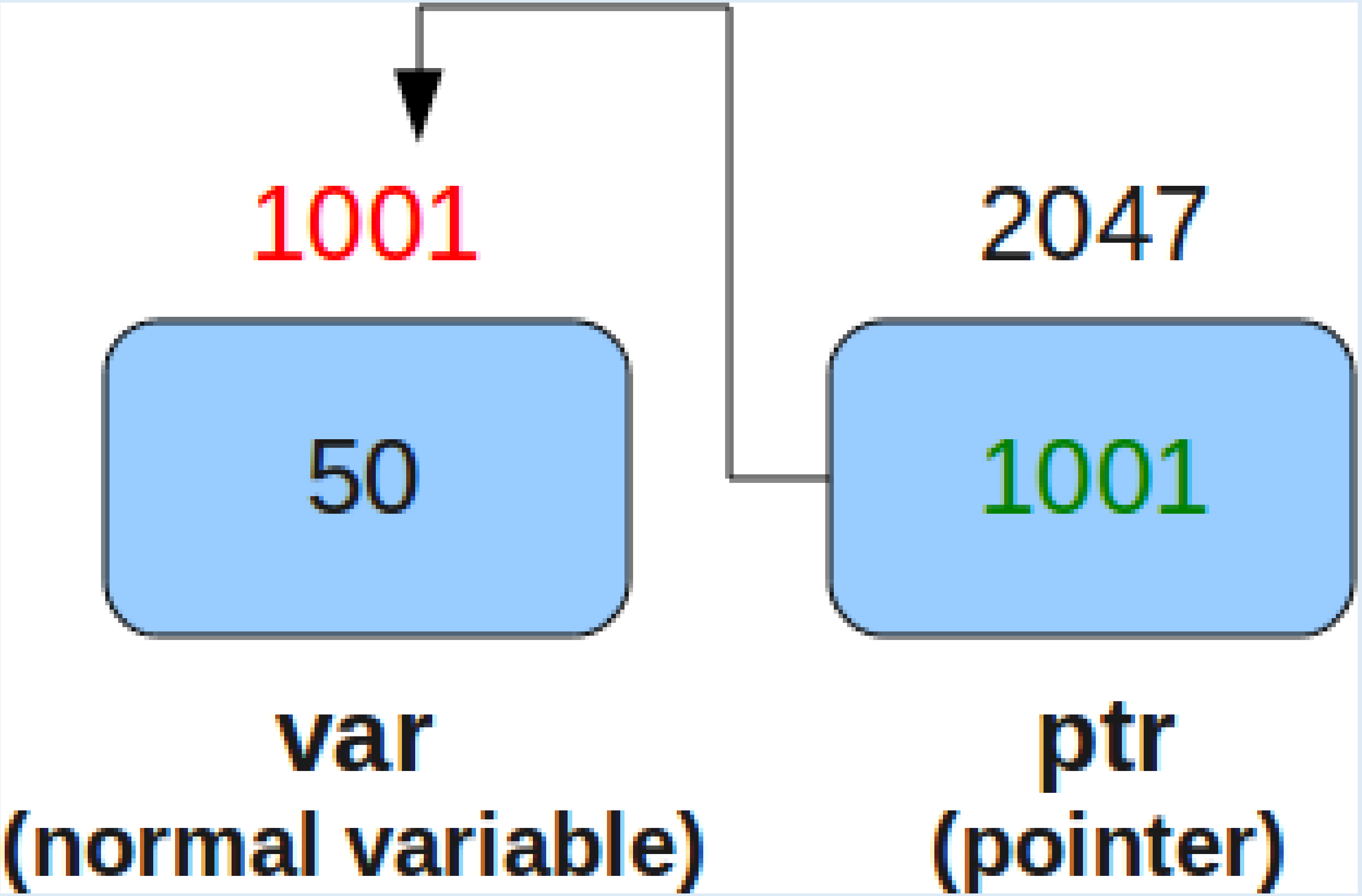


What is a pointer?

The variable that stores the address of another variable is called a pointer.



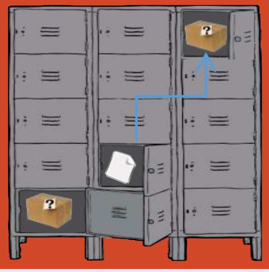
Pointers Vs Normal Variables

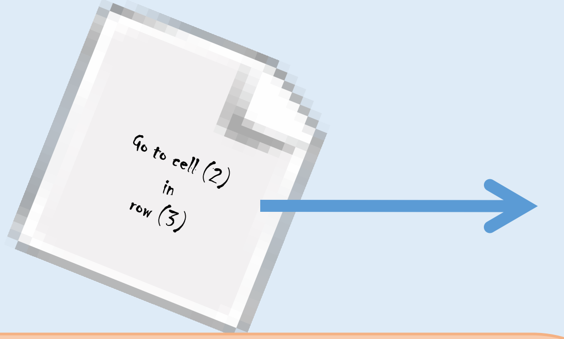


Pointer Declaration

```
dataType * pointerVarName;
```

Pointer Declaration





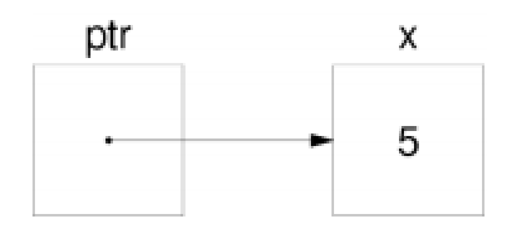
```
int var1;

float var2;

char var3;
```

```
int * var1Ptr;

float * var2Ptr;

char * var3Ptr;
```

Assignment of Pointer Variables

The address of a variable can be obtained by preceding the name of a variable with an ampersand sign (&), known as address-of operator. For example:

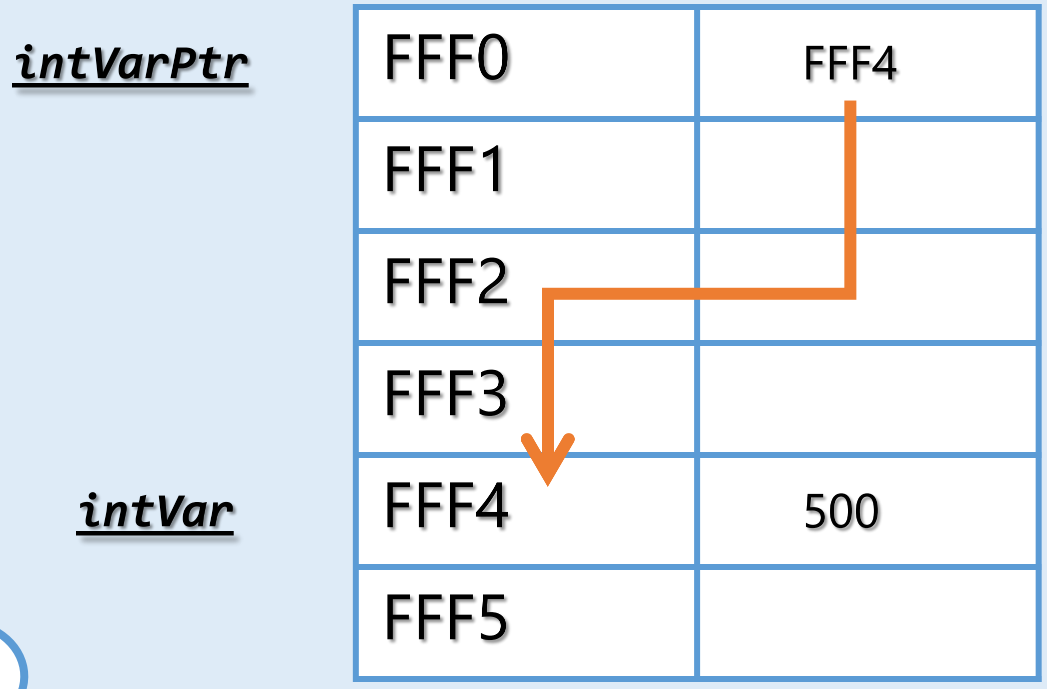
```
int  x = 5;
int  *x_ptr;
x_ptr = &x;
```



Assignment of Pointer Variables

```
int intVar = 500;

int * intVarPtr;

intVarPtr = &intVar;

cout<< intVarPtr;
```

Holds the address



intVarPtr	FFF0	FFF4
	FFF1	
	FFF2	
	FFF3	
intVar	FFF4	500
	FFF5	

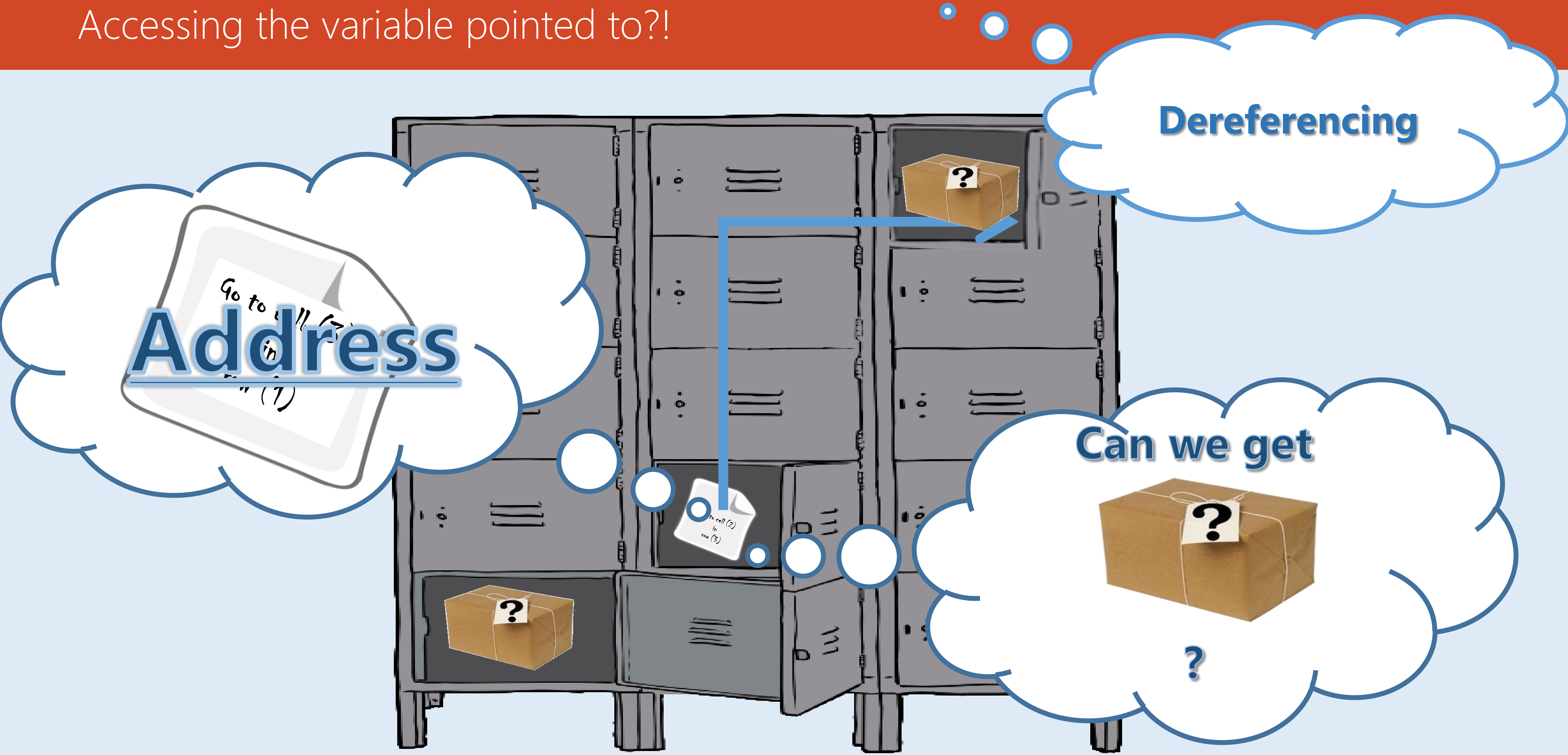
Assignment of Pointer Variables

Same types!!

```
#include <iostream>
using namespace std;

int main()
{
        int intVar = 500;
        float floatVar = 450.5;

        int * intVarPtr;

        intVarPtr = &floatVar;

        cout<< intVarPtr ;
}
```

```
intVarPtr = &intVar;

float * floatVarPtr    =
&floatPtr;
```

Accessing the variable pointed to?!



Address

Dereferencing

Can we get

?

Dereference operator (*)

- As just seen, a variable which stores the address of another variable is called a pointer. Pointers are said to "point to" the variable whose address they store.

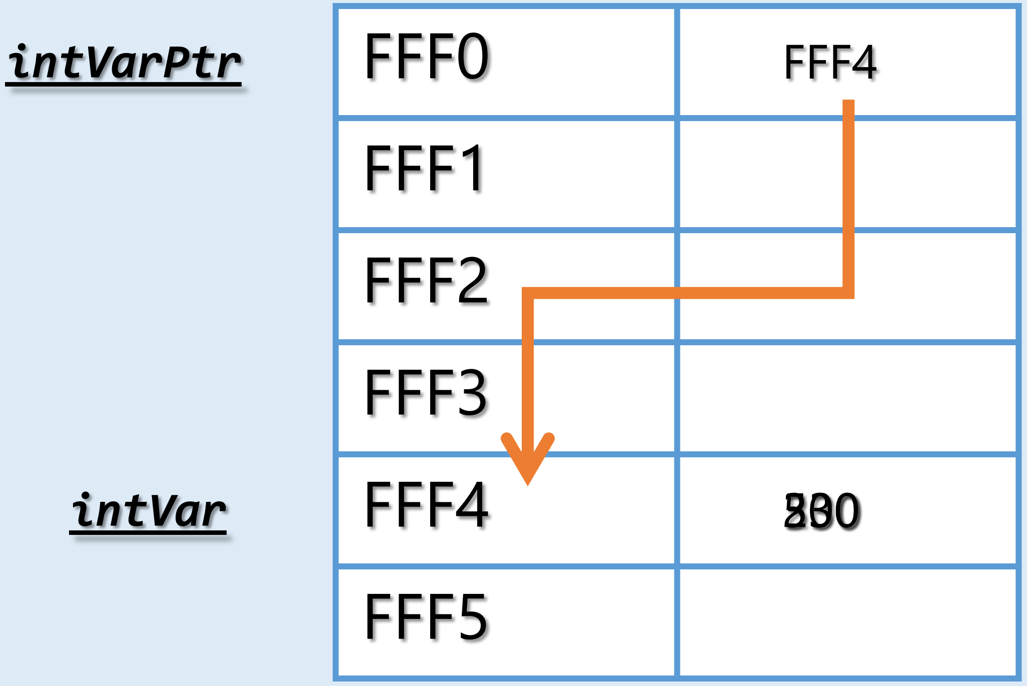
- An interesting property of pointers is that they can be used to access the variable they point to directly.

- This is done by preceding the pointer name with the dereference operator (*).

Accessing the variable pointed to?!

Dereferencing

```
int intVar = 500;

int * intVarPtr = &intVar;

cout<< * intVarPtr;                → 500

* intVarPtr = 230;

cout<< * intVarPtr;                → 230

( * intVarPtr) += 100;

cout<< * intVarPtr;                → 330
```



intVarPtr	FFF0	FFF4
	FFF1	
	FFF2	
	FFF3	
intVar	FFF4	500 230 330
	FFF5	

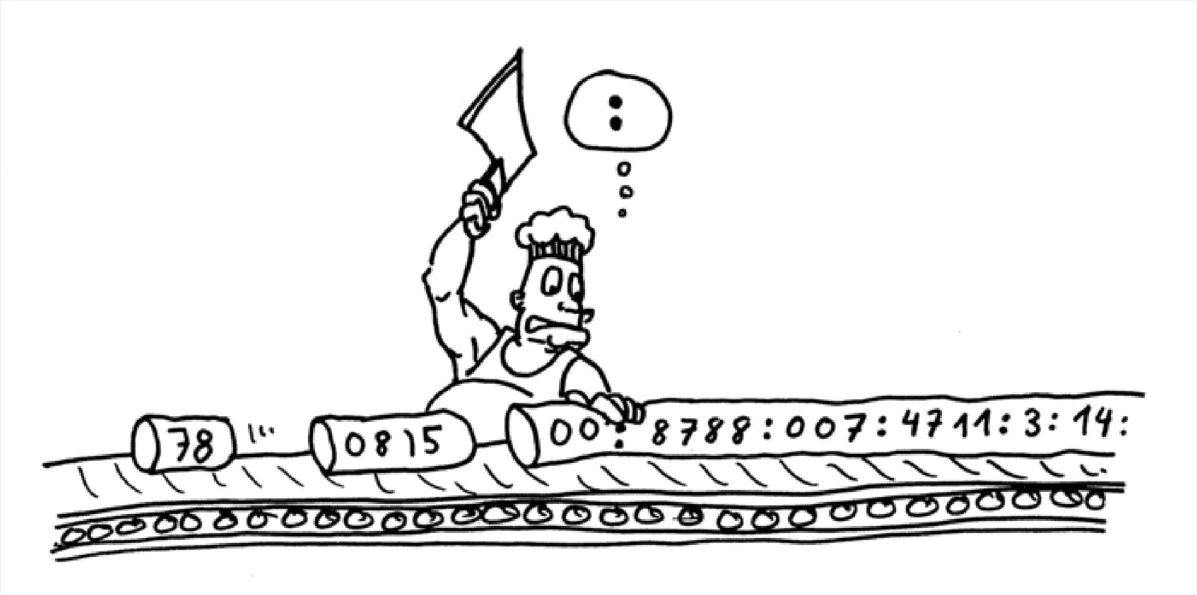
Remember ☺☺

Asterisk (*) can be used in 2 different operations

1. Pointers declaration → int* ptr

2. Dereferencing operation → cout << *ptr

Exercises!

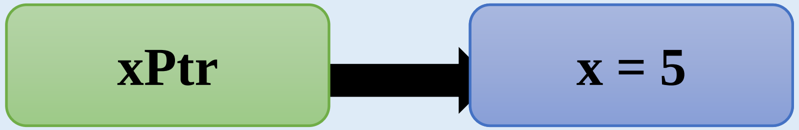


Exercise #1

- Trace the following code segment:

```
int x = 7;
int *xPtr = &x;
cout<<&x <<endl;
cout<<xPtr<<endl;
cout<< x<<endl;
cout<<*xPtr<<endl;
*xPtr = 5;
cout<<x<<endl;
```

Assume that x is saved at address 5000 in memory



Result:

```
5000
5000
7
7
5
```

Exercise #2

Trace the following code segment:

```
int *p;

int i;

int k;

i = 42;

k = i;

P = & i;
```

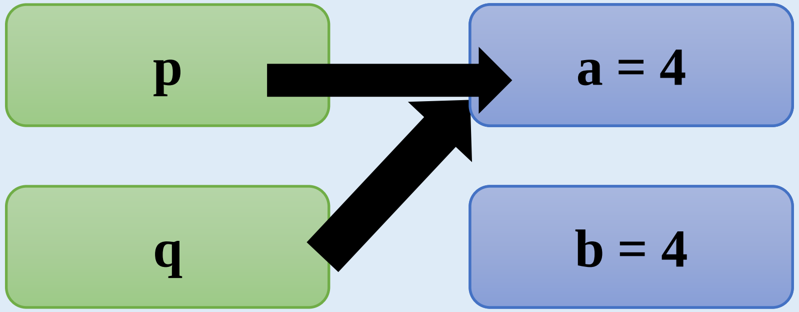
After these statements, which of the following statements will change the value of i to 75?

a. k = 75;
b. *k = 75;
c. p = 75;
d. *p = 75;
e. Two or more of the answers will change i to 75.

Exercise #3

Trace the following code segment:

```
int a; int b;
int* p; int* q;
a = 3;
p = &a;
q = p;
b = 4;
*q = b;
cout << *p << a;
```



Result:

4 4

Dynamic Allocation

New & Delete (Dynamic Allocation)

Static

Compile time

```
int x;
x = 5;
//no delete
```

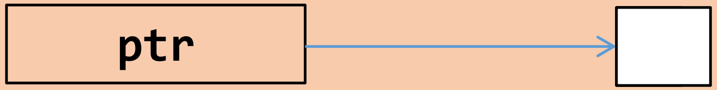
Garbage Collector

Dynamic

Run Time

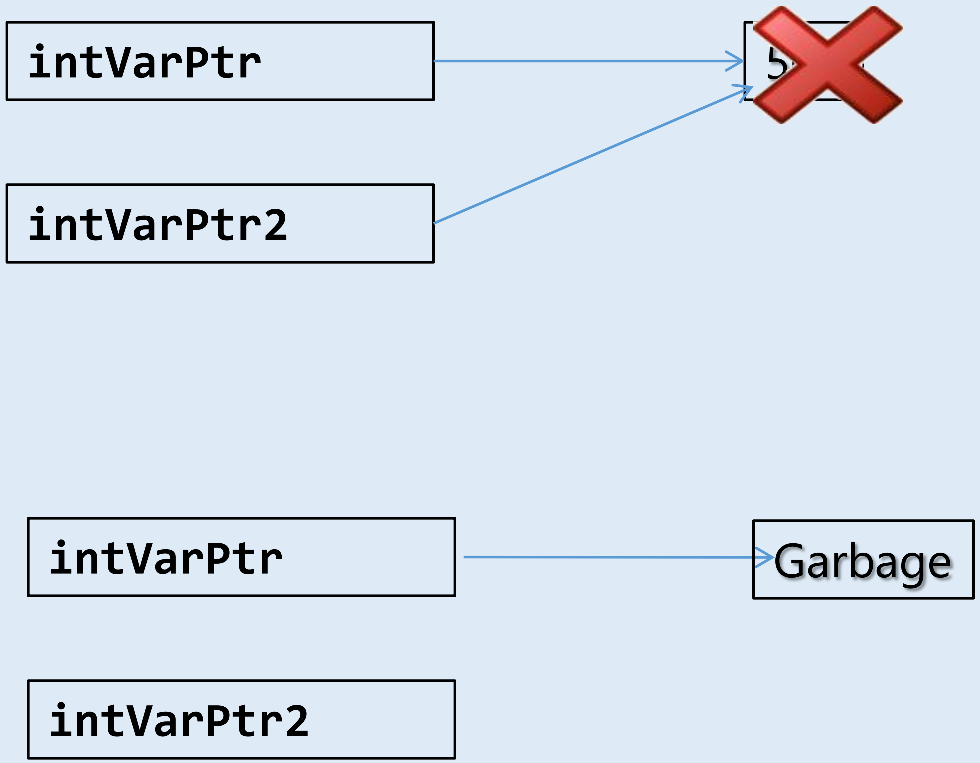
Use Pointers !!

```
int *ptr = new int;
 *ptr = 5;
delete ptr;
ptr = NULL;
```



Example

```
int * intVarPtr;

intVarPtr = new int;

*intVarPtr = 500;

int * intVarPtr2 = intVarPtr;

delete intVarPtr2;

cout<<*intVarPtr<<endl;
```



intVarPtr

intVarPtr2

5

intVarPtr

Garbage

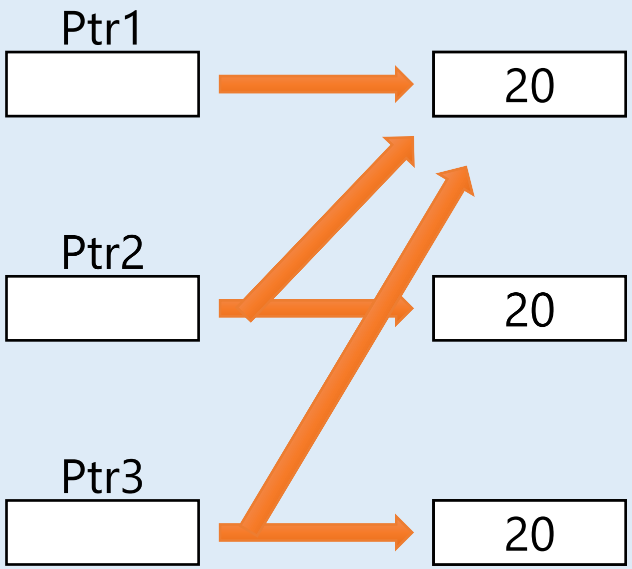
intVarPtr2

Exercise #4

```
int *ptr1 = new int;

*ptr1 = 10;

int *ptr2 = new int;

*ptr2 = 20;

int *ptr3 = new int;

(*ptr3) = (*ptr1)+(*ptr2);

cout<<*ptr1<< " " <<*ptr2<< " " <<*ptr3<<endl;

ptr2 = ptr1;

(*ptr3) = (*ptr1)+(*ptr2);

cout<<*ptr1<< " " <<*ptr2<< " " <<*ptr3<<endl;

ptr3 = ptr1;

(*ptr3) = (*ptr1)+(*ptr2);

cout<<*ptr1<< " " <<*ptr2<< " " <<*ptr3<<endl;

delete ptr1;
```

Dangling Pointer & Memory leaks

```
10  20  30
10  10  20
20  20  20
```



Thank you!