PIC 10A

Lecture 23: Pointers

Pointers

Recall that while we think of variables by their names like:

```
int numbers;
```

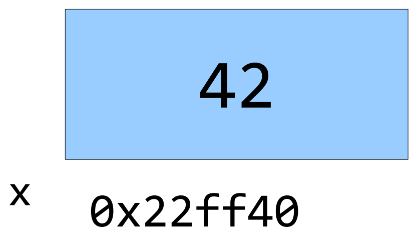
Computer likes to think of variables by their memory address:

0012FED4

A pointer is a variable that stores the address of another variable.

Memory address of a variable

Lets say we have an int variable x:

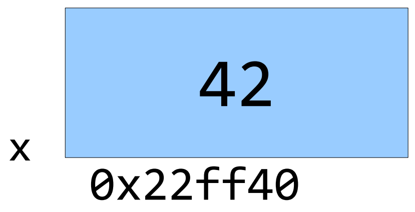


The variables name is x and that is how programmer refers to it in his program.

The variable is stored in memory location 0x22ff40. We say that x has address 0x22ff40.

The variable is storing the value 42.

Memory address of a variable



If I want to get an address of a variable I can use the address operator &.

&x is the address of the variable x.

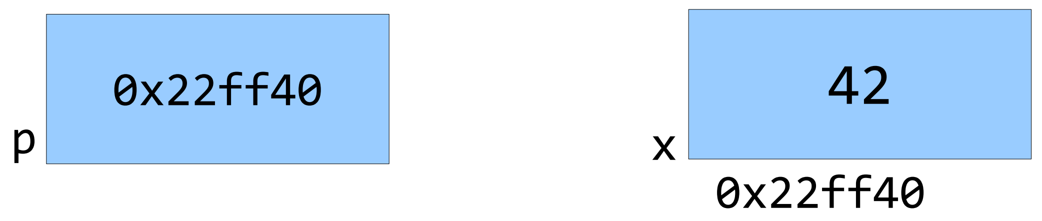
Where have we seen this before?

```
cout << x;
cout << &x;
```

What is the output?

Pointers

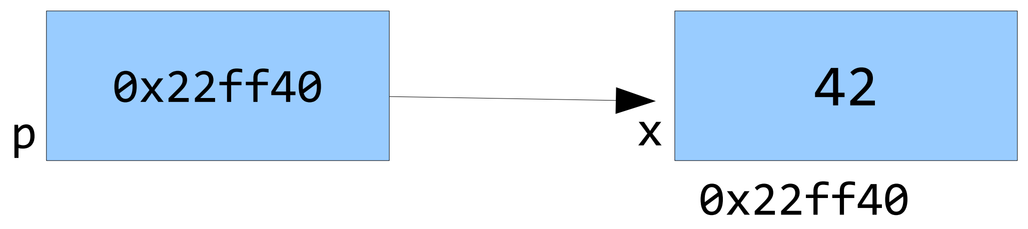
Lets say we have an int pointer p and a regular int variable x.



Here the pointer p is storing the address of the variable x.

We say that p points to x.

We draw an arrow to indicate that p points to x.



Declaring pointers

Syntax for declaring a pointer:

```
type * name;
```

Example:

```
int * p; // Declares an integer pointer, but it is
         // uninitialized.

int a=2;

p = &a;  // p now points to a.
```

Draw a picture

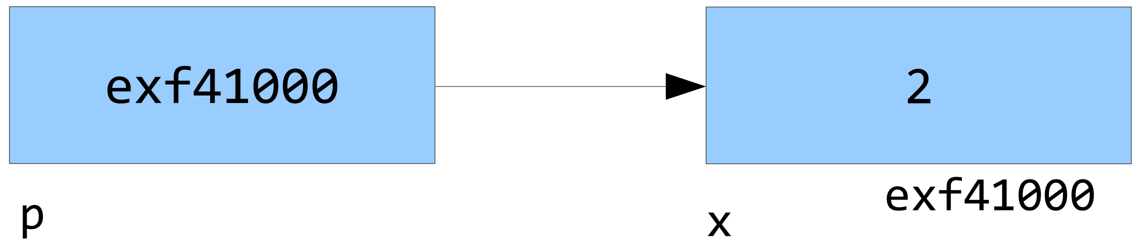
Dereferencing operator *

We can access the memory location that a pointer points to by using a dereferencing operator.

We can use the dereferencing to output the contents of the memory location:

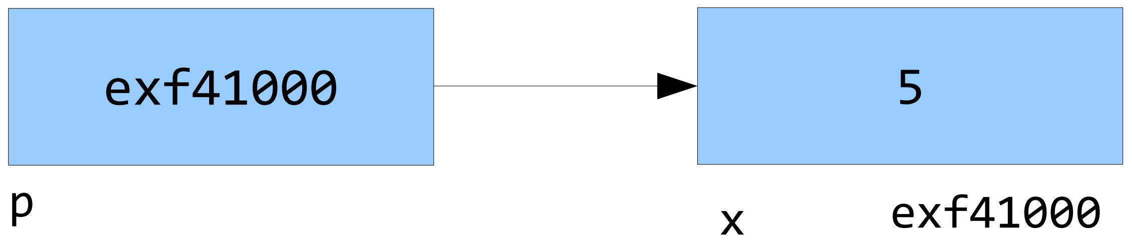
```
cout << *p; // Outputs the value 2.
```

Dereferencing operator *



We can even use the dereferencing operator to change the value stored at the memory location where p points to!!

```
cout << x;
*p = 5;
cout << *p;
cout << x;
```



So *p acts exactly like the variable x that p points to.

Pointer fun

```
int a=2;
int *p; // Do not confuse this * with dereferencing
p = &a; // p points to a
```

What is the output of the following lines?

```
cout << *p;
cout << p;
cout << &a;
cout << &p;
cout << *&a;
```

Pointers to different kinds of variables

```
double *b;
string *MyString;
Product *ProductPointer;
```

Remember that the pointer type has to match the variable type.

I can not do:

```
double a;
int *p;

p=&a; // p can only point to int variable!
```

Example

```
int a;
int *p;

p=&a;
a=23;

cout << *p;
a=56;
cout << *p;
*p=5;
cout <<a;
```

Example

It is very important to initialize pointers before using them.

```
int *q;
cout << *q;   // crashes the program if q is not
              // pointing to anything.
```

Understanding difference between addresses and reference

Be careful when declaring multiple ptrs you must put * on every one of them :

```
int *a, b;  // a is int*, but b is an int

int * p1, *p2;
*p1=7;
*p2=8;
```

This is wrong why?

This even compiles but remember uninitialized pointers are just like unintialized variables. The pointers point to some random memory.

It's not a good idea to change what is stored in random memory location...Your own program might be stored there!!

Understanding difference between addresses and reference

```
int * p1, *p2;
int a,b;

p1=&a;
p2=&b;

*p1=7;
*p2=8;
```

Lets draw a picture

Example

```
int *p1, *p2;
int a=7,b=8;
p1=&a;
p2=&b;
```

Draw a picture

```
p1=p2;
```

Draw a picture

```
p1=&a;
p2=&b;
*p1 = *p2;
```

Draw a picture

Example

```
int *p1, *p2;
int a=7,b=8;
p1=&a;
p2=&b;

p1=p2;

*p1 = 10;

cout << *p2 << " " << b;
```

Example

```
int *p1, *p2;
int a=7,b=8;
p1=&a;
p2=&b;

int *temp;

temp=p1;
p1=p2;
p2=temp;

cout << a << " " << b << " " << *p1 << " " << *p2;
```

new

If I write:

```
int *p;
```

I have a pointer but it does not yet point to anything.

```
p = new int;
```

This declares some free memory to which p now points to.

```
*p = 17;  // This new memory now holds the value 17.
```

Note this new memory has no name. It is only accessible through p. Through p we have created a variable.

new

Often when you are programming you do not know how many objects you will need.

We can create these objects as we go using dynamic memory allocation.

For example we can write:

```
Product * p;

p = new Product;
```

You might say why not just write:

```
Product my_product;
```

Heap vs stack memory

Difference is that objects created with a new command live in heap memory and objects created with ordinary declarations live in the stack memory.

When a variable is stored in stack memory it lives in a storage area that is associated with a function in which the variable was declared:

```
void foo()
{
Product my_product;
....
}
```

When computer exits `foo`, `my_product` automatically dies.

Heap vs stack memory

However, when we write

```
Product * p;
p = new Product;
```

This memory allocated for the Product is now in heap memory. This Product can only be accessed through the pointer p, but it will stay alive until the programmer explicitly deallocates the memory.

So one reason to use pointers is to be able to create objects with “longer life span”.

Delete

To free this memory for other applications I should do:

```
delete p;
```

Important: p is not deleted. Just the memory where p pointed to has been released for other applications.

We can still use the pointer p:

```
int x = 3;
p = &x;
```

When p is no longer pointing to anything useful set it to NULL.

```
delete p;
p=NULL;
```

NULL is defined in <cstdlib>

Memory management

A program will crash if memory runs out!

The delete function allows us to free up space in memory, so we don’t run over the memory limit.

Suppose we have a class Database that stores a lot of data and takes up a lot of memory.

```
Database *p = new Database;

... Code...

delete p;
```

Now the memory is freed.

Dangling pointers

A common run-time error is to use a pointer that doesn't point anywhere.

Two dangling pointer errors:
Using a pointer that was not initialized.
Using a pointer that was deleted.

```
int *p;
cout << *p;

p = new int;
delete p;
cout << *p;
```

In both cases, we do not get a compile error. Here p just points to a random spot, usually containing a garbage value.

Example with pointers

```
void fun (int *p, int *&q)
{
    *p = 2;
    q = p;
    int c=7;
    p=&c;
    return;
}

int main()
{
    int x=5, y=10;
    int *a = &x;
    int *b = &y;
    fun(a, b);
    return 0;
}
```

Arggh pointers seem pointless!

Where is all of this going?

What good are these things that seem so complicated?

Will we actually ever use these things for anything?

In PIC 10B you will use pointers a lot and start to understand their power.

Next time I will outline a couple of examples where pointers are the best way to solve a problem.