Computer Science 172
Chapter 11
Introduction to Pointers

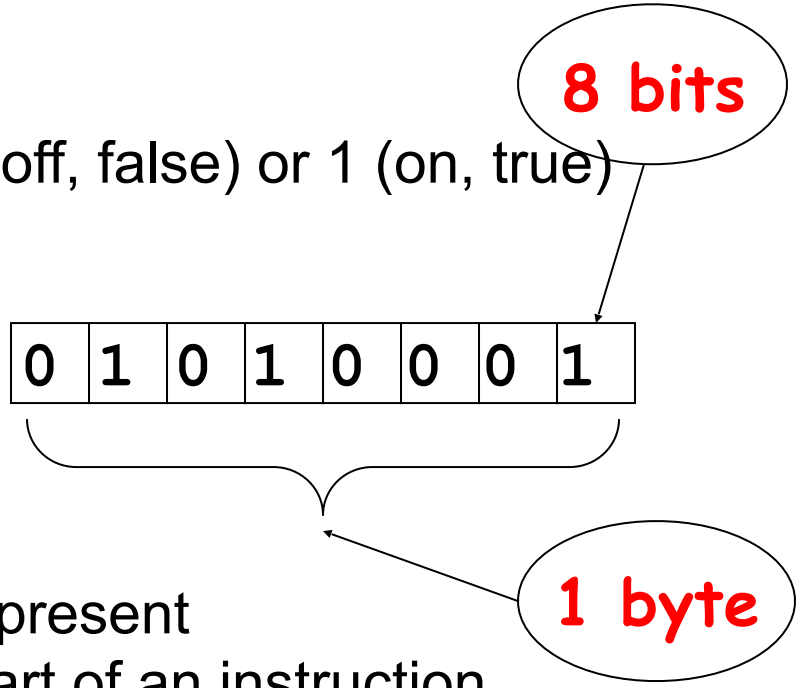
Course Goal: Increase foundational computer science concepts including: object-oriented programming, **pointers, dynamic memory**, templates, file I/O, characters, strings and recursion

Today's Agenda

- Pointer Basics (11.2)
- Using **const** with pointers (11.4)
- Pointers and Arrays (11.5)

Computers store information in memory

- **Bit** - A Binary digIT
  - Smallest piece of memory
  - Can store one of two values: 0 (off, false) or 1 (on, true)



- **Byte**
  - Is 8 consecutive bits that can represent a character OR a number OR part of an instruction
  - Each byte has an **address**

Each **byte** location in memory has a unique **address**

ADDRESSES	MEMORY	
: :	: :	
198	0000001	
199	0000000	
200	0000000	
201	0000010	
202	1001000	H
203	1000101	E
204	1001100	L
205	1001100	L
206	1001111	O
: :	: :	

4 bytes

ASCII Data

C++ allows us to use variable names instead of **physical memory addresses**!

NOTE: Some variables require more storage space than others!

Memory Address	Actual Memory	Variable Names & Size
10	2010	`int year;` (requires 4 bytes)
14	190.50	`double salary;` (requires 8 bytes)
22		`Person michael;` (requires 20 bytes)
42		
43		
44		

To "get" the physical memory address of a variable, use the & operator

```
#include <iostream>
using namespace std;
int main()
{
        int num = -23;
        cout << "Address of num: " << &num;
}
```

The & operator returns the **address** of the local variable num.

Practice

```
int num = -23;

cout << "Number of bytes: " << sizeof(num) <<
endl;
cout << "Address of num: 0x" << &num << endl;
cout << "Address of num: " << long(&num) << endl;
```

Typecast the **memory address** to an int.

Put this in main().
Add other variables of different types
Make a new class, and add a variable of that type

You can create a **POINTER VARIABLE** to **"store" the address** of another variable

```
int *  ptr;
```

Read this pointer variable declaration as:
"ptr stores the address of an int"
OR **"ptr is a pointer to an int"**
OR **"ptr is a pointer"**

- **NOTE**: Spacing in the declaration does not matter:

```
int* ptr;
int *ptr;
```

It is considered good practice to always initialize your pointer variables with **NULL**!

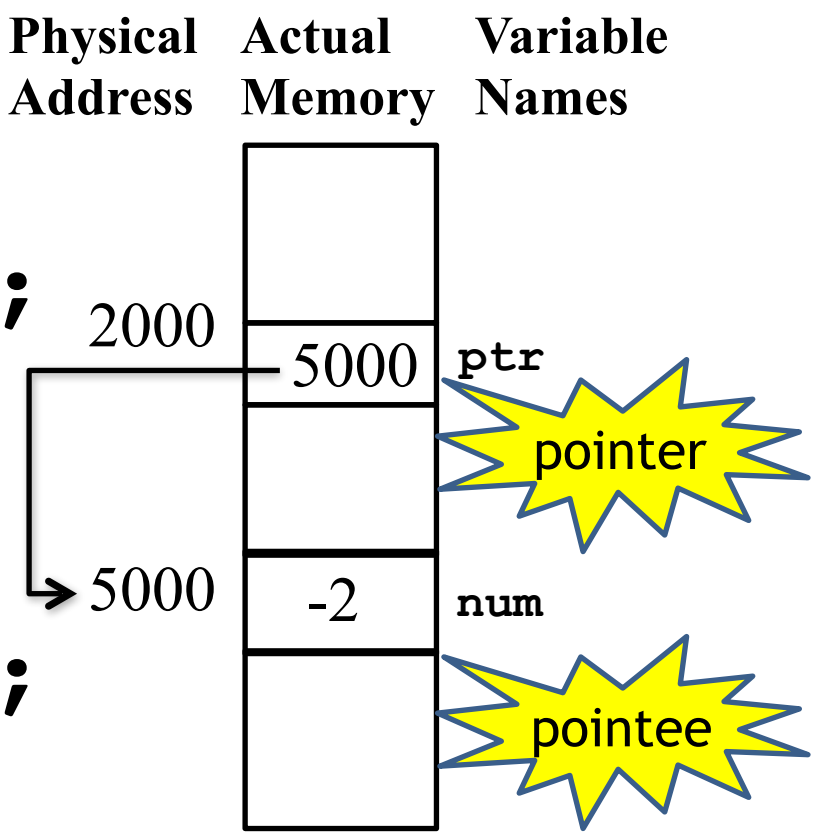
```
int* ptr = NULL;
```

You can then **"store" the address** of a variable in a **pointer variable**

```
int num = -2;
int * ptr = NULL;
ptr = &num;
```

OR

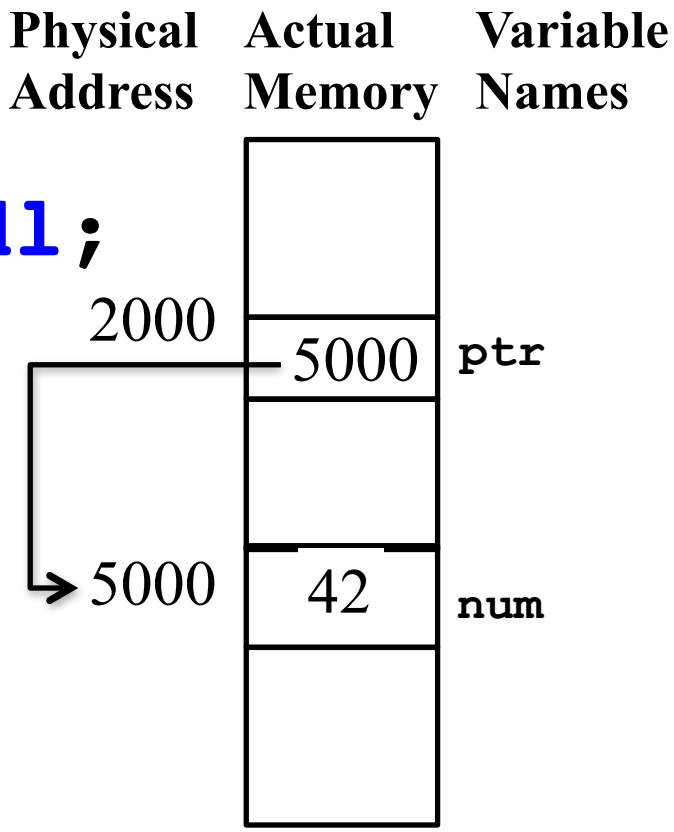
```
int * ptr = &num;
```



You can **"access"** the **memory** that a **pointer variable** is referring to with the * **operator**

```
cout << *ptr << endl;
```


```
*ptr = 42;
```



You can use relational operators to compare the **pointer** or the **memory address** that the pointer refers to

```
if (ptr1 == ptr2)
```

This == operator will return true IF ptr1 and ptr2 each contain the same address.

```
if (*ptr1 == *ptr2)
```

This == operator will return true IF the <u>contents</u> of the "dereferenced" pointers are the same

Pointer type definition MUST match the type of object you are pointing too!

```
double cost;
int *iptr = &cost; // won't work
double *fptr = &cost; // OK!
```

Because **cost** is a **double**, any pointer variable for **cost** must be a **double***

In-class Exercise
Convert this summation to use p rather than n;

```
#include <iostream>
using namespace std;
int main()
{
    int n;
    int *p = &n;
    cout << "Gimme a number (10-50): ";
    cin >> n;

    int sum = 0;
    for (int i = 1; i <= n; i++) {
        sum += i;
    }

    cout << "summation of " << n << " = " << sum << endl;
}
```

const and pointers

- You can make the **pointer constant**

```
int X = -2, Y = -3;
int * const ptr = & X;
ptr = &Y; // NOT OK!  Constant pointer
```

const and pointers

- You can specify that it is illegal to change the contents of the memory location that the pointer refers to
  - Useful with functions that take pointers as parameters (late)

```
int X = -2;
const int * ptr = & X;
*ptr = -3; // NOT OK!  Constant pointer
```

const and pointer

- You can combine const to get both

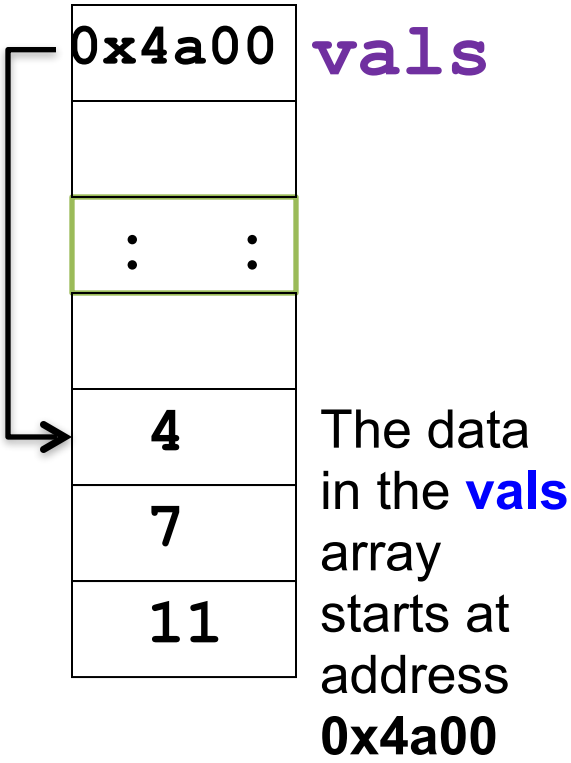
```
int X = -2, Y = -3;
const int * const ptr = & X;
ptr = &Y; // NOT OK!
*ptr = -3; // NOT OK!
```

Pointers and Arrays

In C++ an **array name** is actually a const **pointer variable** (that **points** to the memory allocated for the array)

```
int vals[3] = {4, 7, 11};
cout << vals[0];
cout << *vals;

int othervals[10];
//illegal, vals is const
vals = othervals;
```



Arrays and pointers have a close relationship in C++

- You can use an **array name** like a **pointer**

```
int vals[3] = { 4, 7, 11 };
cout << *vals;     // displays 4
```

- You can use a **pointer** like an **array name**

```
int *  valptr = vals;
cout << valptr[1]; // displays 7
```

How would you use pointers to output the second value in the array?

Pointer Arithmetic

Subscript Arithmetic

Given:

```
int vals[3]={4,7,11};
int* valptr = vals;
cout << *valptr; // shows 4
cout << *(valptr+2); // shows
11
```

What does *(**valptr + 2**) mean?

It means access the integer stored at the address:

(address of valptr) + (2*sizeof(int))

Pointer Arithmetic

```
int vals[3]={4,7,17};
int * valptr = vals;
```

Array elements can be accessed :

Array access method	Example
array name and [ ]	vals[2] = 17;
pointer to array and [ ]	valptr[2] = 17;
array name and subscript arithmetic	*(vals+2) = 17;
pointer to array and subscript arithmetic	*(valptr+2) = 17;

You can use ++ and -- with pointer variables

- Assume the variable definitions:

```
int vals[3]={4,7,11};
int *valptr = vals;
```

- Examples of use of **++** and **--**

```
valptr++; // points at 7
valptr--; // now points at 4

vals++;     //again, illegal
```

Hands On Exercise

Using ++ to traverse an array with a pointer variable

1. Start with the code given below. Evaluate what the program will do.
2. Compile and run the program. Did it do what you thought it would do? How does it work?

```
#include <iostream>
using namespace std;
int main()
{   const int SIZE = 4;
    int vals[SIZE]={4,7,11,25};
    int *ptr = vals;
    for ( ptr=vals; ptr < vals + SIZE; ptr++ ) {
        cout << *ptr << " is stored at";
        cout << " address 0x" << ptr << " (" << int(ptr) << ")";
        cout << " in memory" << endl;
    }
}
```

Use this as a starting point
Write a loop that sums the values in an array, using pointers

More Pointer Arithmetic

- Assume the variable definitions:

```
int vals[]={4,7,11};
int *valptr = vals;
```

- Examples

```
valptr = vals; // points at 4
valptr += 2;   // points at 11
cout << valptr - val; // prints 2
```