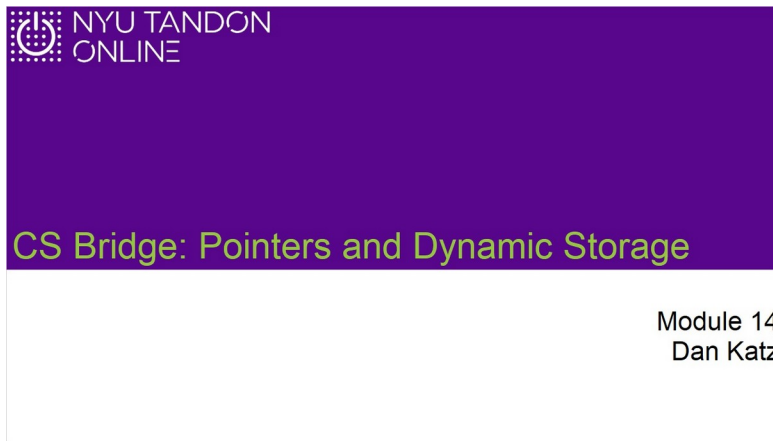


CS Bridge Module 11 Pointers and Dynamic Storage

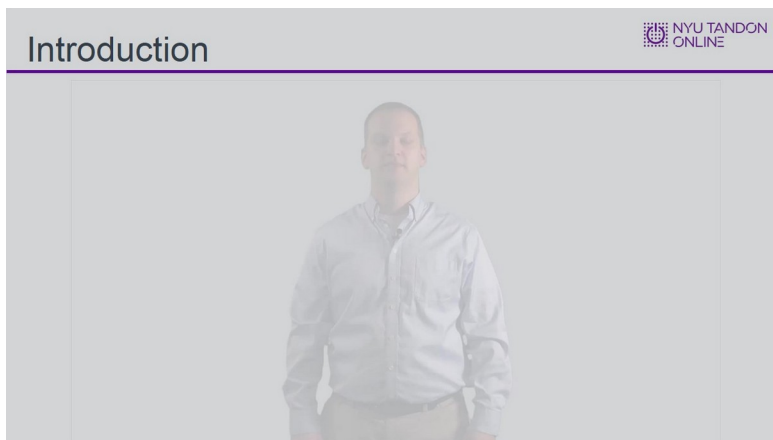
1. Pointers and Dynamic Storage

1.1 Title Slide



Notes:

1.2 Introduction



1.3 Pointers – Why?

Pointers – Why?



- A way to store a “reference” to an object
- A way to store large amounts of information not associated with a function (the heap)
- A way to link objects together



1.4 What it looks like

What It Looks Like



- In C++ a pointer must specify the datatype that it points to
- In main memory, the pointer is simply a stored number
- The stored number represents the memory address of the item being pointed to

int * ptr;



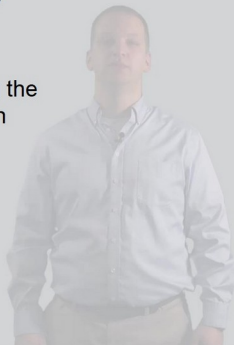
1.5 Getting pointers to point

Getting Pointers to Point



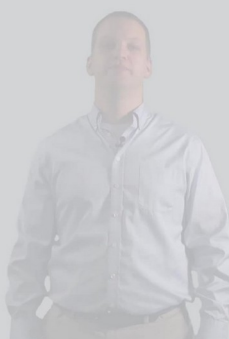
- Pointers can be made to point to something
- The address of the item being pointed to is stored in the pointer.
- We can get the address of an item by using the “address-of” operator, the ampersand “&”, in front of the variable’s name.

```
int x = 100;  
int * ptr;  
ptr = &x;
```



1.6 Accessing data from a pointer

Accessing Data From a Pointer



- Pointers would be pretty useless if all they could do was point.
- We can access the data that a pointer is pointing to by "dereferencing" the pointer. We do this using the dereferencing operator, the star.

```
int x = 100;
int * ptr;
ptr = &x;
cout << *ptr;
*ptr = 20;
```

1.7 Knowledge Check

(Multiple Choice, 10 points, 1 attempt permitted)

Knowledge Check

In the following code, what is the value of x after the second cout?


```
int x = 100;
int *ptr;
ptr = &x;
cout << *ptr;
*ptr = 20;
x = 40;
cout << *ptr;
```

☐ 20
☒ 40
☐ 100

Correct	Choice	Feedback
	20	Note that the value of x was changed twice!
X	40	Good work!
	100	Note that the value of x has changed twice!

20 (Slide Layer)

Knowledge Check



In the following code, what is the value of x after the second cout?

```
int x = 10;
int*ptr;
prt = &x;
cout<<*ptr;
*ptr=20;
x=40;
cout<<*ptr;
```


Incorrect

Note that the value of x was changed twice!

Continue

40 (Slide Layer)

Knowledge Check



In the following code, what is the value of x after the second cout?

```
int x = 10;
int*ptr;
prt = &x;
cout<<*ptr;
*ptr=20;
x=40;
cout<<*ptr;
```


Correct

Good work!

Continue

100 (Slide Layer)

Knowledge Check



In the following code, what is the value of x after the second cout?

```
int x = 10;
int*ptr;
prt = &x;
cout<<*ptr;
*ptr=20;
x=40;
cout<<*ptr;
```

Incorrect

Note that the value of x has changed twice!

Continue

1.8 In the code below, what does the “&” symbol represent?

(Multiple Choice, 10 points, 4 attempts permitted)

Knowledge Check

NYU TANDON
ONLINE

In the code below, what does the “&” symbol represent?

```
int x = 100;  
int * ptr;  
ptr = &x;  
cout << *ptr;
```

☐ Pass by Reference

☒ Address

☐ AND operator

☐ String Concatenation

Correct	Choice	Feedback
	Pass by Reference	While this is true for function parameters, it has a different meaning here!
X	Address	Correct!
	AND operator	The AND operator is represented by “&&”
	String Concatenation	String concatenation can be done with the “+” symbol

Pass by Reference (Slide Layer)

Knowledge Check

NYU TANDON
ONLINE

In the code below, what does the "&" symbol represent?

```
int x = 100;
int * ptr;
ptr = &x;
cout << *ptr;
```

Incorrect

While this is true for function parameters, it has a different meaning here!

Continue

Address (Slide Layer)

Knowledge Check

NYU TANDON
ONLINE

In the code below, what does the "&" symbol represent?

```
int x = 100;
int * ptr;
ptr = &x;
cout << *ptr;
```

Correct

Correct!

Continue

AND operator (Slide Layer)

Knowledge Check

NYU TANDON
ONLINE

In the code below, what does the "&" symbol represent?

```
int x = 100;
int * ptr;
ptr = &x;
cout << *ptr;
```

Incorrect

The AND operator is represented by "&&"

Continue

String Concatenation (Slide Layer)

Knowledge Check

In the code below, what does the "&" symbol represent?

```
int x = 100;
int * ptr;
ptr = &x;
cout << *ptr;
```

Incorrect

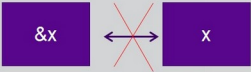
String concatenation can be done with the "+" symbol

Continue

1.9 What if a pointer doesn't point to anything?

What if a pointer doesn't point to anything?

- Pointers always point to something
- If the pointer isn't pointing to something VALID, it should point to:
 - NULL or nullptr**



1.10 Defining multiple pointers

Defining Multiple Pointers

- When defining pointers, the * is associated with only the next item in the sequence. If you need multiple pointers in the same line, you need more stars!

```
int *ptr1, x;
int *ptr2, *ptr3;
```

1.11 Lets get dynamic!

Lets Get Dynamic!



- Pointers wouldn't be much use if they could only point to objects that were created already!
- Pointers can point to heap-dynamic memory!
- Heap-Dynamic memory is allocated when you ask for it, and remains allocated until you destroy it.
- This means it can survive function calls!
- You have to be very careful to destroy it after you're done, it's not destroyed automatically.

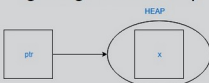


1.12 Well, that's new

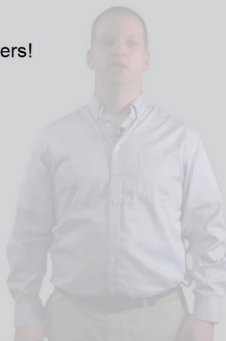
Well, That's New



- A variable can be created on the heap for a particular datatype, but it does not have a name.
- If it doesn't have a name, how do you refer to it? ...pointers!
- Once memory is allocated on the heap, it will not be deallocated until you do it, or the program ends.
- If you lose track of a heap-dynamic variable, it becomes "garbage on the heap" aka, a "memory-leak."



```
int * ptr = new int;
```

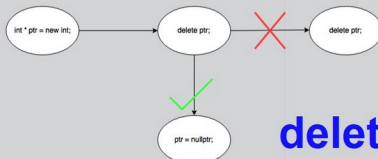


1.13 For every new, there must be delete

For every new, there must be delete



- If memory is allocated, it must be destroyed
- You must destroy it once and only once
 - Forgetting to destroy it results in a memory leak
 - Deleting it twice results in a "double delete," which will crash your program
- Deleting NULL or nullptr has no effect



```
delete ptr;
```



1.14 Knowledge Check

(Multiple Choice, 10 points, 1 attempt permitted)

Knowledge Check

NYU TANDON
ONLINE

What should be done immediately after deleting a pointer?

☐ Check if it's deleted

☒ Set pointer to nullptr

☐ Try to access the pointer

☐ Delete the pointer again

Correct	Choice	Feedback
	Check if it's deleted	When a pointer is deleted, the pointer still exists. This is another step to take to ensure that there isn't a memory leak
X	Set pointer to nullptr	Correct!
	Try to access the pointer	Once a pointer is deleted, there is no way to access it
	Delete the pointer again	Deleting a pointer after it has already been deleted will crash the program!

Check if it's deleted (Slide Layer)

Knowledge Check

NYU TANDON
ONLINE

What should be done immediately after deleting a pointer?

Incorrect

When a pointer is deleted, the pointer still exists. This is another step to take to ensure that there isn't a memory leak

Continue

☐ Check if it

☒ Set pointer

☐ Try to access the pointer

☐ Delete the pointer again

Set pointer to nullptr (Slide Layer)

Knowledge Check

NYU TANDON
ONLINE

What should be done immediately after deleting a pointer?

Correct

Correct!

Continue

☐ Check if it

☒ Set pointer

☐ Try to access the pointer

☐ Delete the pointer again

Delete the pointer again (Slide Layer)

Knowledge Check

NYU TANDON
ONLINE

What should be done immediately after deleting a pointer?

Incorrect

Deleting a pointer after it has already been deleted will crash the program!

Continue

☐ Check if it

☒ Set pointer

☐ Try to access the pointer

☐ Delete the pointer again

Try to access the pointer (Slide Layer)

Knowledge Check

NYU TANDON
ONLINE

What should be done immediately after deleting a pointer?

Incorrect

Once a pointer is deleted, there is no way to access it

Continue

☐ Check if it

☒ Set pointer

☐ Try to access the pointer

☐ Delete the pointer again


1.15 What about arrays?

What About Arrays?

NYU TANDON
ONLINE

- Heap-dynamic arrays don't have to have a static size!
- Once created, heap-dynamic arrays can't grow, but they can be created to be any size initially.
- Since we are working with a pointer, a new, larger, heap-dynamic array can be made to replace a smaller one!
- Deleting a heap-dynamic array is a bit different

```
int x;  
cout << "How big?";  
cin >> x;  
int *arr = new int[x];  
delete [] arr;
```




1.16 What can we do with heap-dynamic arrays

What can we do with heap-dynamic arrays

NYU TANDON
ONLINE


- Heap-dynamic arrays are no different than the "normal" arrays which you've been using.
- Use heap-dynamic arrays just like all other arrays, using the square-brackets operator.
- You can also use pointer arithmetic...




1.17 Pointer arithmetic

Pointer Arithmetic

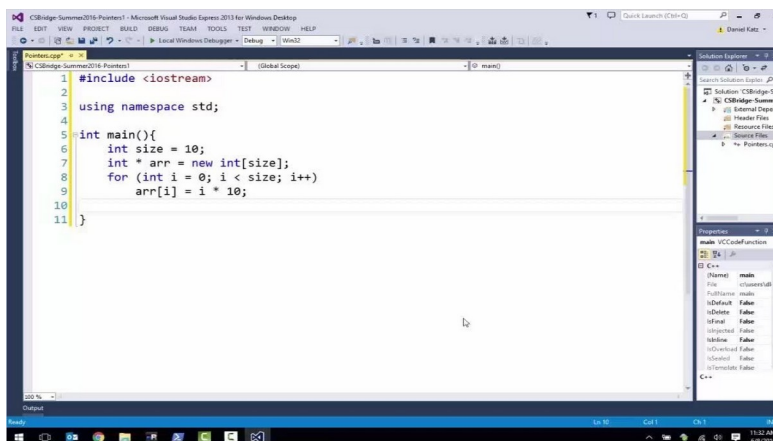
- C++ allows pointers to be manipulated using simple math operations.
- Use the addition operator ("+") to move the pointer forward some number of spaces ($x = x + 5$ will make the pointer point to 5 elements higher in the array)
- Use the subtraction operator to move the pointer backwards
- Use ++ or -- as you'd like.





Daniel Katz

1.18 A real-example of a growing array



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

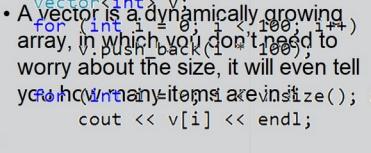
int main(){
    int size = 10;
    int * arr = new int[size];
    for (int i = 0; i < size; i++)
        arr[i] = i * 10;
}
```


1.19 Someone's done this already

Someone's Done This Already

Of course, C++ has something called the "Standard Template Library."

- The STL contains a data-type known as a "vector."
- A vector is a dynamically growing array, in which you don't need to worry about the size, it will even tell you how many items are in it.






Daniel Katz

Notes:

1.20 But wait... there's more!

But Wait... There's More!




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

int main()
{
    vector<int> v;
    for (int i = 0; i < 100; i++)
        v.push_back(i * 100);


    for (int i: v)
        cout << i << endl;
}
```


- The syntax is a bit strange (it was borrowed from another language and brought into C++ later)




1.21 Conclusion

Conclusion





1.22 End of Module



NYU TANDON
ONLINE

End of Module

Exit