Chapter 9 Pointers

Tuesday, November 28, 2017 12:48 PM

9.1 Getting the Address Of a Variable
9.2 Pointer Variables
9.3 The Relationship Between Arrays and pointers
9.4 Pointer Arithmetic
9.5 Initializing Pointers
9.6 Comparing pointers
9.7 pointers as Function parameters
9.8 Focus on Software Engineering: Dynamic Memory Allocation
9.9 Focus on Software Engineering: Returning pointers from Functions
9.10 Using Smart Pointers to Avoid Memory Leaks
9.11 (Optional) Focus on problem Solving and program Design: A Case Study

9.1 Getting the Address of a Variable

Variable - specific location in memory (RAM)
Each variable in program is stored at a unique address

Getting the address of a variable
int num = -99;
cout << &num; // prints address in hexadecimal

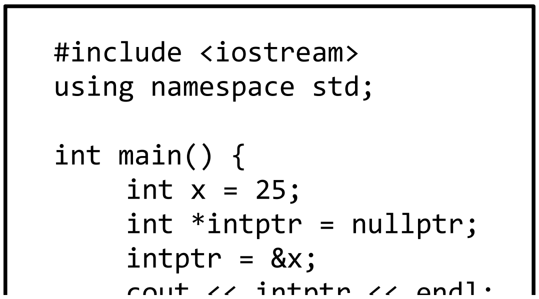
9.2 Pointer Variables

- Pointer Variable - variable that holds an address
  Aka "pointer"

- Define a pointer variable
  int* intptr;

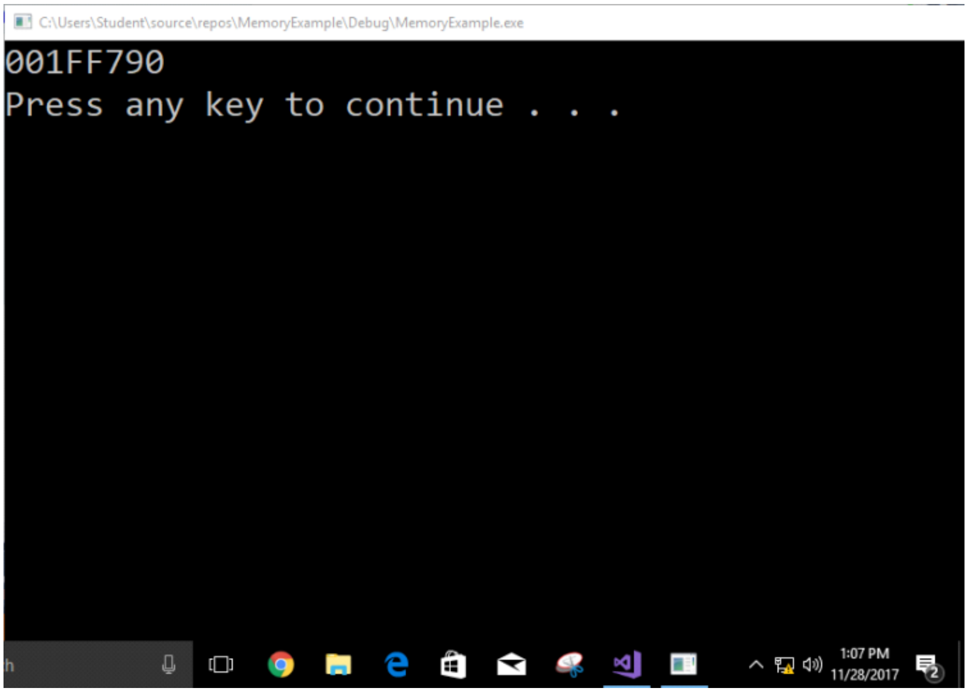
  Intptr can hold the address of an int

  int *intptr; //same as above
  int * intptr; //same as above



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

int main() {
     int x = 25;
     int *intptr = nullptr;
     intptr = &x;
     cout << intptr << endl;
```

```
        int x = 25;
        int *intptr = nullptr;
        intptr = &x;
        cout << intptr << endl;
        system("pause");
        return 0;
    }
```


C:\Users\Student\source\repos\MemoryExample\Debug\MemoryExample.exe
1:07 PM
11/28/2017

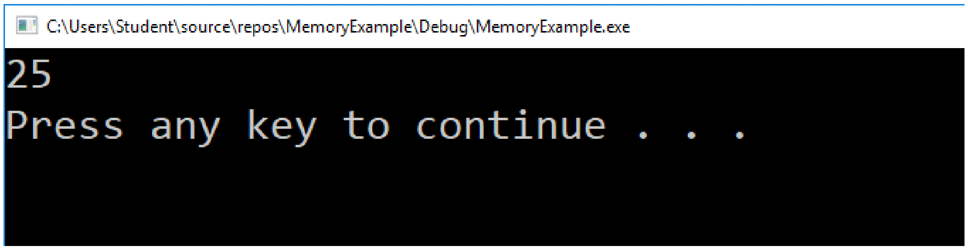
```
001FF790
Press any key to continue . . .
```

The Indirection Operator (*)

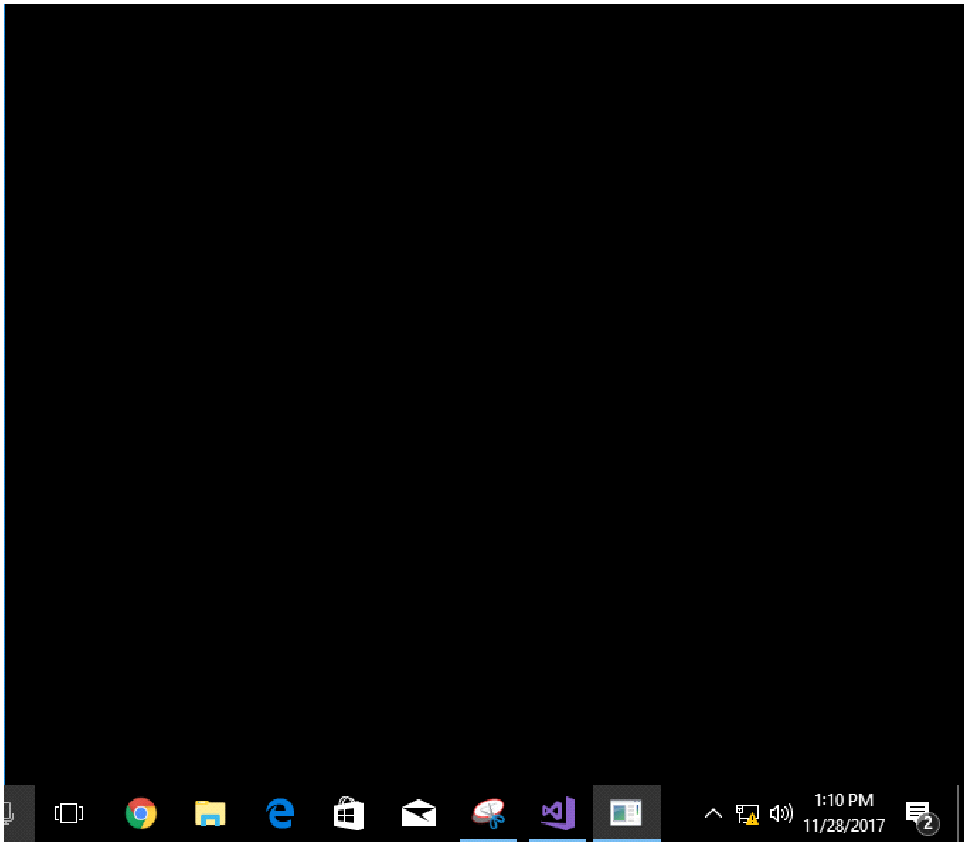
* dereferences a pointer

- It allows you to access the item that the pointer points to.

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

int main() {
    int x = 25;
    int *intptr = nullptr;
    intptr = &x;
    cout << *intptr << endl;
    system("pause");
    return 0;
}
```


C:\Users\Student\source\repos\MemoryExample\Debug\MemoryExample.exe

```
25
Press any key to continue . . .
```



9.3 The Relationship Between Arrays and Pointers

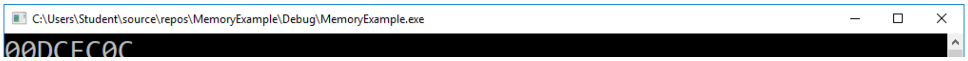
- Array name is starting address of array

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

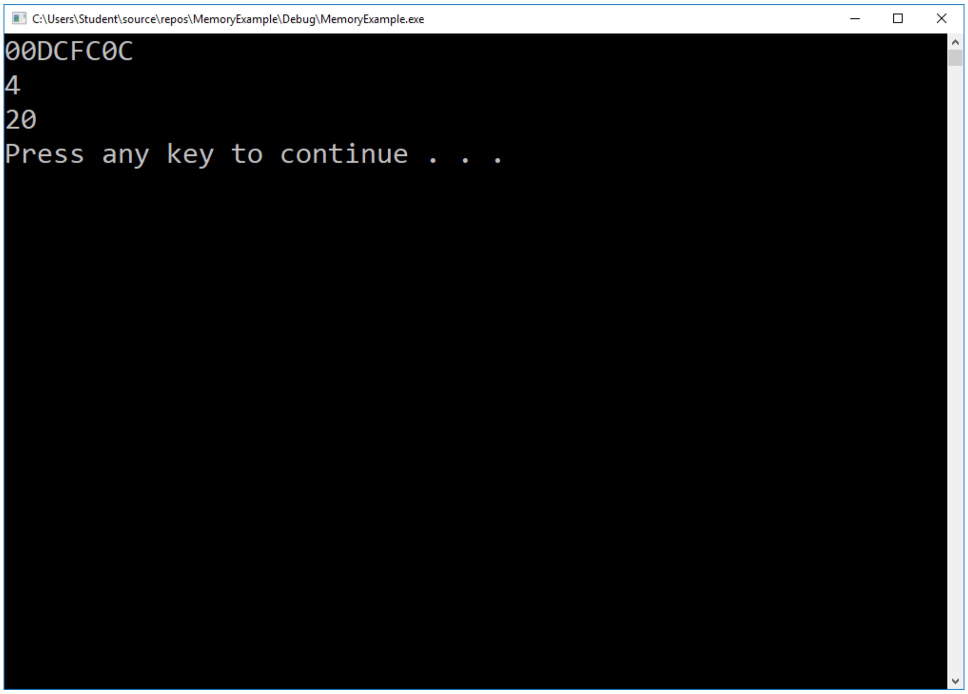
4	7	11

cout << vals << endl;

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

int main() {
     int vals[] = { 4, 7, 11 };
     cout << vals << endl; // prints address of first
                                // element in vals
     cout << (*vals) << endl; // prints 4
     cout << 5*(*vals) << endl; //prints 20
     system("pause");
     return 0;
}
```


C:\Users\Student\source\repos\MemoryExample\Debug\MemoryExample.exe

```
00DCEC0C
```

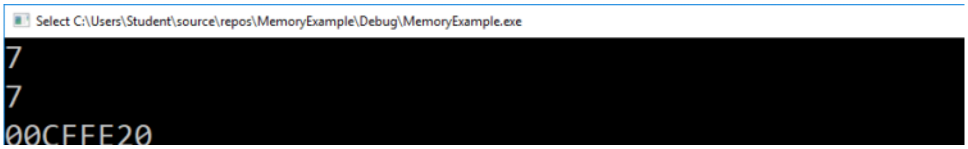

C:\Users\Student\source\repos\MemoryExample\Debug\MemoryExample.exe

```
00DCFC0C
4
20
Press any key to continue . . .
```

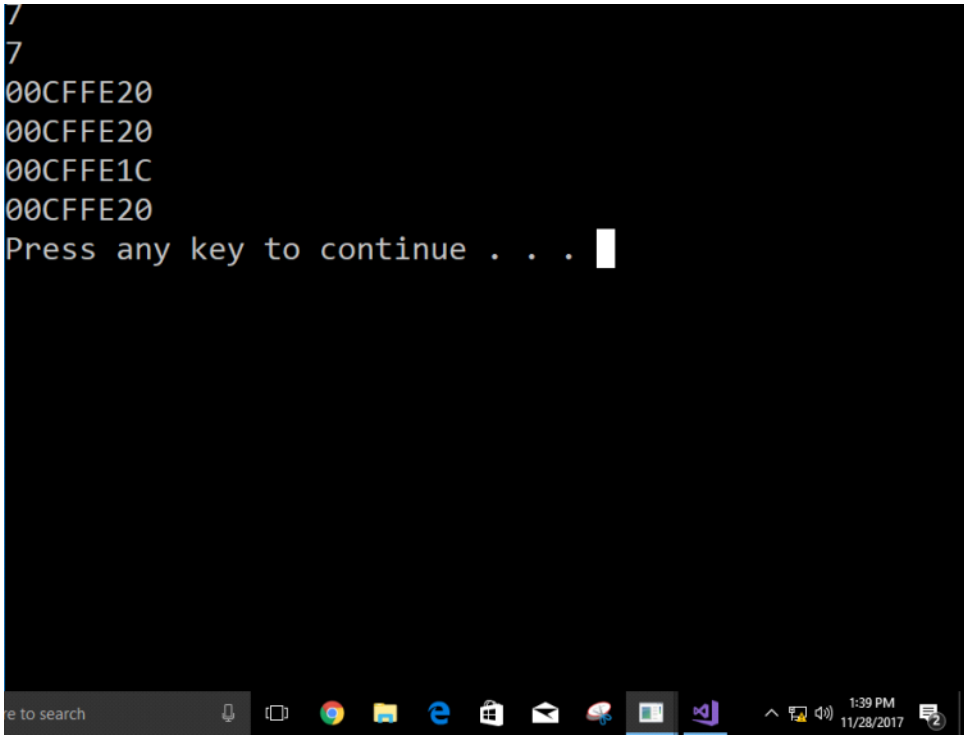
Pointer can be used as an array name:
int[] vals = {4, 7, 11};

4	7	11

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

int main() {
     int vals[] = { 4, 7, 11 };
     int *valptr = vals;
     cout << vals[1] << endl; //prints 7
     cout << valptr[1] << endl; //prints 7

     cout << &vals[1] << endl; //prints address of the 7
     cout << &valptr[1] << endl; //prints address of the 7

     cout << valptr << endl; // prints the address of the 4
     cout << valptr + 1 << endl; //prints the address of the 7
     cout << *(valptr + 2) << endl; //prints the 11

     system("pause");
     return 0;
}
```


Select C:\Users\Student\source\repos\MemoryExample\Debug\MemoryExample.exe

```
7
7
00CFFE20
```



```
7
7
00CFFE20
00CFFE20
00CFFE1C
00CFFE20
Press any key to continue . . .
```

Array Access

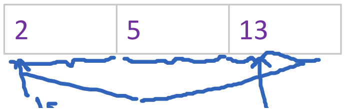
```
int vals[] = { 4, 7, 11 };
int *valptr = vals;
```

array name and []	vals[2]	11
pointer to array and []	valptr[2]	11
array name and subscript arithmetic	*(vals + 2)	11
Pointer to array and subscript arithmetic	*(valptr + 2)	11

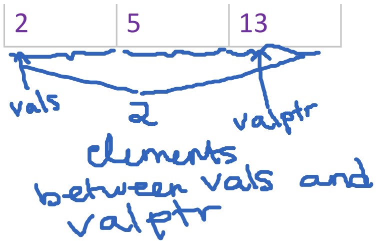
Conversion:
vals[i] <--------> *(vals + i)

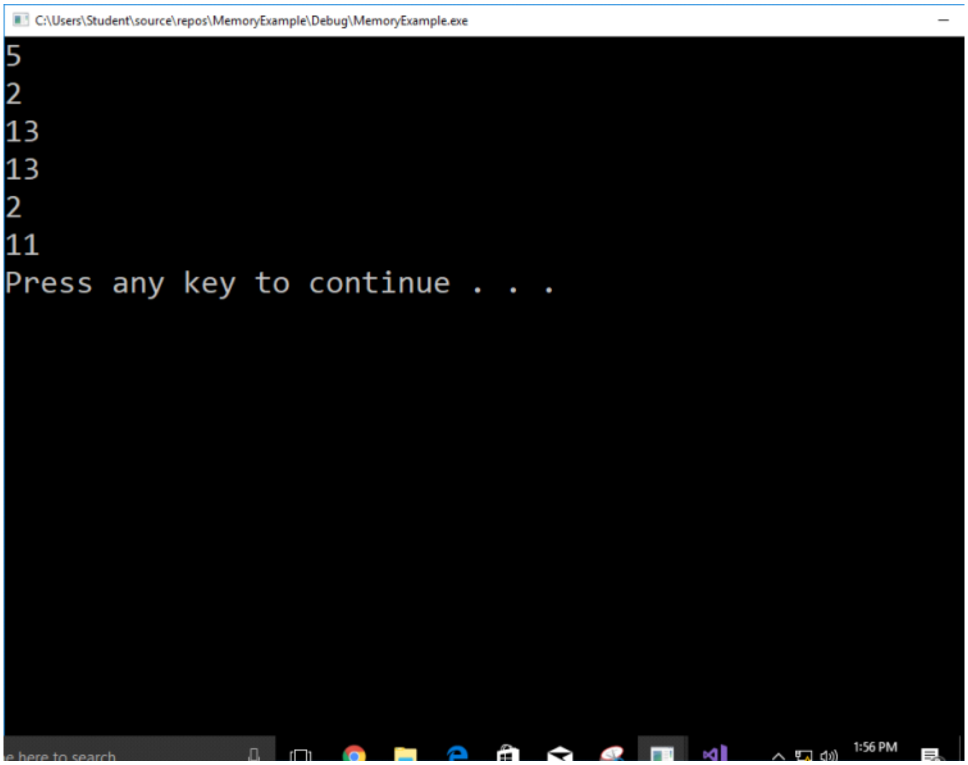
9.4 Pointer Arithmetic

```
    int vals[] = { 2, 5, 13 };
    int *valptr = vals;

    valptr++; //points at 5
    cout << *valptr; // prints a 5

    valptr--; // points at 2
    cout << *valptr; // prints a 2

    cout << *(valptr + 2); //prints a 13

    valptr = vals; //points at 2
    valptr += 2; //points at 13
    cout << *valptr; //prints a 13

    cout << valptr - vals; //prints number
                 //of ints between
                 //valptr and vals
```



```
cout << valptr - vals; //prints number
               //of ints between
               //valptr and vals


cout << *valptr - *vals;
```




C:\Users\Student\source\repos\MemoryExample\Debug\MemoryExample.exe
1:56 PM

```
5
2
13
13
2
11
Press any key to continue . . .
```

9.5 Initializing Pointers

int num, *numptr = &num;
int val[3], *valptr = val;

Cannot mix data type:
double cost;
int *ptr = &cost; // won't work
double *ptr = &cost; // will work

9.6 Comparing pointers

- Relational Operators can be used to compare pointers
  - In that case, the addresses which the pointers store are compared

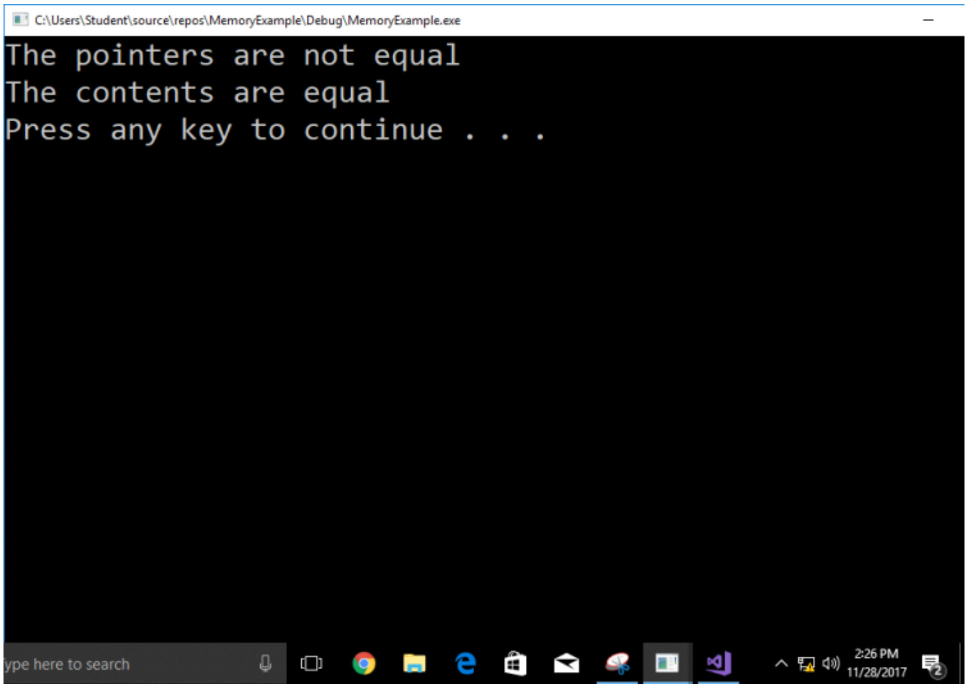
>
<
>=
<=
==

- Comparing addresses *in* pointers is not the same as comparing the contents *pointed at by* pointers.

```
#include <iostream>
```

comparing the contents *pointed at by* pointers.

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

int main() {
     int a = 25;
     int b = 25;
     int *ptr1 = &a;
     int *ptr2 = &b;

     if (ptr1 == ptr2)
          cout << "The pointers are equal" << endl;
     else
          cout << "The pointers are not equal" << endl;
     if (*ptr1 == *ptr2)
          cout << "The contents are equal" << endl;
     else
          cout << "The contents are not equal" << endl;


     system("pause");
     return 0;
}
```


C:\Users\Student\source\repos\MemoryExample\Debug\MemoryExample.exe
ype here to search
2:26 PM
11/28/2017

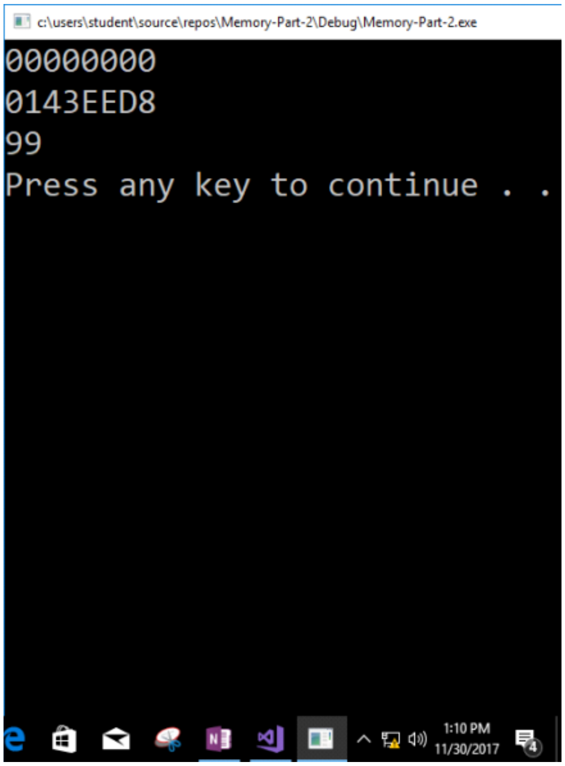
```
The pointers are not equal
The contents are equal
Press any key to continue . . .
```

9.8 Dynamic Memory Allocation

- Storage for a variable can be allocated while program is running
- The new operator
  - Allocates memory
  - Returns address of the newly created memory location

- ○ Allocates memory
- ○ Returns address of the newly created memory location

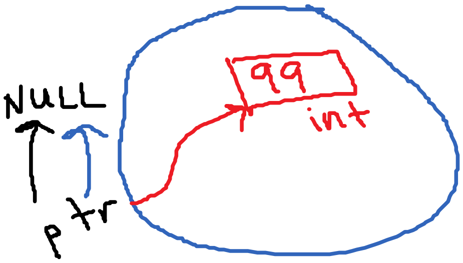
```
#include <iostream>
using namespace std;

int main(){

    int *ptr = nullptr;
    cout << ptr << endl;
    ptr = new int;
    *ptr = 99;
    cout << ptr << endl;
    cout << *ptr << endl;

    system("pause");
    return 0;
}
```


c:\users\student\source\repos\Memory-Part-2\Debug\Memory-Part-2.exe

```
00000000
0143EED8
99
Press any key to continue . .
```

Releasing Dynamic Memory

Use delete to free dynamic memory

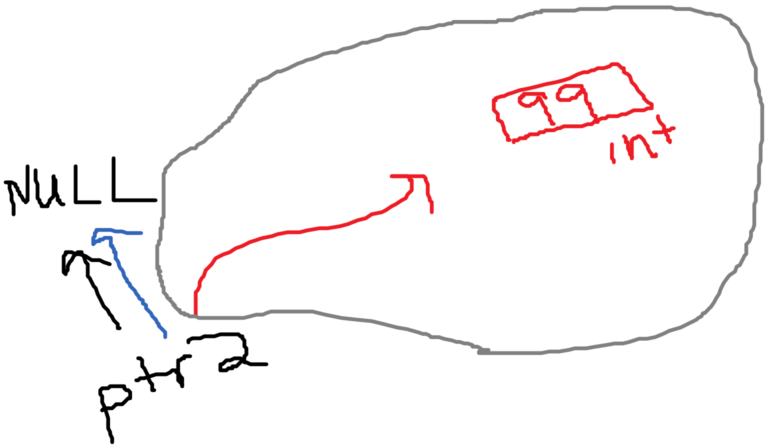


```
#include <iostream>
using namespace std;

int main() {

    int *ptr = nullptr;
    cout << ptr << endl;
    ptr = new int;
```

```
	int *ptr = nullptr;
	cout << ptr << endl;
	ptr = new int;
	*ptr = 99;
	cout << ptr << endl;
	cout << *ptr << endl;
	delete ptr;
	ptr = nullptr;

	system("pause");
	return 0;
}
```

00000000
01385380
99

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

int main() {

	int *ptr2 = nullptr;
	cout << ptr << endl;
	ptr2 = new int;
	*ptr2 = 99;
	cout << ptr << endl;
	cout << *ptr << endl;
	ptr2 = nullptr; //BAD, MEMORY LEAK
	delete ptr2; // HAS NO EFFECT


	system("pause");
	return 0;
}
```



00000000
01385380
99

Memory Leaks

- new creates a storage location in memory
- If all the pointers to that storage location in memory get reassigned,
  - There is no way to refer to that location in memory anymore

Memory leak - a storage location in memory that no longer has any pointers to it

Dynamic memory can also be allocated with arrays:

- Use [] to delete arrays

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

int main(){
	const int SIZE = 3;
	int *ptr = nullptr;
	ptr = new int[SIZE]
int main(){
        const int SIZE = 3;
        int *ptr = nullptr;
        ptr = new int[SIZE];

        for (int i = 0; i < SIZE; i++){
                ptr[i] = i * i;
                cout << ptr[i] << endl;
        }

        delete [] ptr;
        ptr = nullptr;
        system("pause");
        return 0;
}


}
```

i
0 1 2
ptr[i] = i*i
0 1 4

0
1
4

9.10 Using Smart Pointers to Avoid Memory Leaks

- In C++11 and C++14, you can use smart pointers to dynamically allocate memory and not worry about deleting the memory when you are finished using it.

- Requires #include<memory>

unique_ptr<int> ptr (new int);

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

int main() {

    int *ptr = nullptr;
    cout << ptr << endl;
    ptr = new int;
    *ptr = 99;
    cout << ptr << endl;
    cout << *ptr << endl;
    delete ptr;
    ptr = nullptr;

    unique_ptr<int> ptr3(new int);
    *ptr3 = 99;
    cout << *ptr3 << endl;
    ptr3 = nullptr;

    system("pause");
    return 0;
}
```

```
    return 0;
}
```

Output:

```
00000000
02955E30
99
99
```

9.7 Pointers as Function Parameters

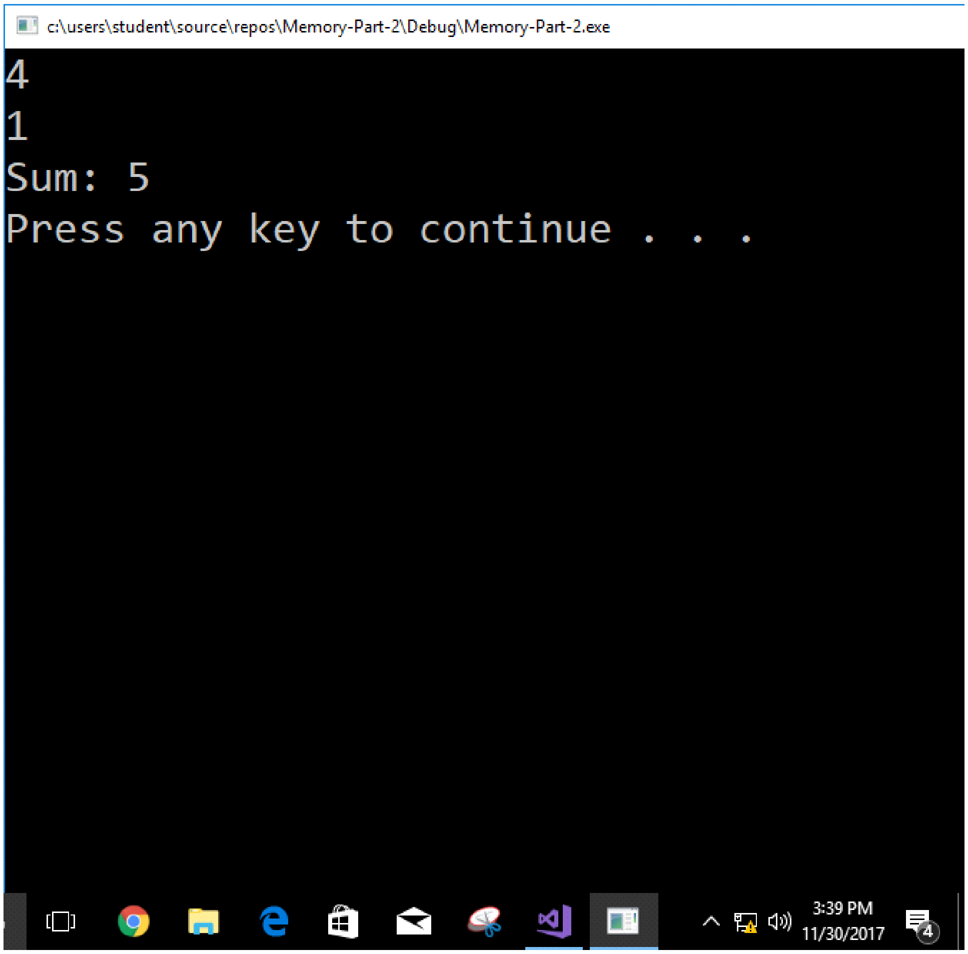
- Functions can accept pointers as arguments.
- Specify the pointer in the parameter list.

```
int sumDice(int *a, int N) {
    int sum = 0;
    for (int i = 0; i < N; i++) {
        sum += *(a + i);
    }
    return sum;
}
```

9.9 Returning pointers from functions

```
int *rollDice(){
    const inst SIZE = 2;
    int *arr = nullptr;

    //Dynamically allocate the array
    arr = new int[SIZE];

    srand(time(0));

    for (int count = 0; count < SIZE; count++){
       arr[count] = rand() % 6 + 1; // generates a random number between 1 and 6
    }
    return arr;
}
```

```
#include <iostream>
#include <cstdlib> //for srand and rand
#include <ctime> // for the time function
using namespace std;

int *rollDice();
int sumDice(int *a, int N);
int *rollDice();
int sumDice(int *a, int N);



int main() {
    int *dice = nullptr;
    dice = rollDice();

    for (int i = 0; i < 2; i++) {
        cout << dice[i] << endl;
    }

    cout << "Sum: " << sumDice(dice, 2) << endl;

    //Free the memory
    delete[] dice;
    dice = nullptr;

    system("pause");
    return 0;
}

int *rollDice() {
    const int SIZE = 2;
    int *arr = nullptr;

    //Dynamically allocate the array
    arr = new int[SIZE];

    srand(time(0));

    for (int count = 0; count < SIZE; count++) {
        arr[count] = rand() % 6 + 1; // generates random number between 1 and 6
    }
    return arr;
}

int sumDice(int *a, int N) {
    int sum = 0;
    for (int i = 0; i < N; i++) {
        sum += *(a + i);
    }
    return sum;
}
```



c:\users\student\source\repos\Memory-Part-2\Debug\Memory-Part-2.exe

```
4
1
Sum: 5
Press any key to continue . . .
```