

Chapter 10 - Pointers

10.1 Pointers and the Address Operator	10. 8 Pointers to Constants and Constant Pointers
10.2 Pointer Variables	10.9 Dynamic Memory Allocation
10.3 The Relationship between Arrays and Pointers	10.10 Returning Pointers from Functions
10.4 Pointer Arithmetic	10.11 Pointers to Class Objects and Structures
10.5 Initializing Pointers	10.12 Selecting Members of Objects
10.6 Comparing Pointers	10.13 Smart Pointers
10.7 Pointers as Function Parameters	

The big idea behind this chapter is

It relates to the previous chapter how ...

The main purpose of this chapter is ...

The key questions are ...

Why:

When:

How:

Why is this material at this point in the class?

You'll know this material when ...

Main assumptions are ...

Opening Thoughts. Write any thoughts or questions you have before reading this material. See if you can find the answers while you read.

Key Ideas. Record major points from the chapter.

[illegible]

List of Figures

1	Address operator demo.	4
2	Difference between regular and pointer variables.	5
3	Demo access different variables with pointers.	6
4	Relationship between arrays and pointers.	6
5	Arrays with pointer notation.	7
6	Demo of reference variables.	7
7	Pointers as function parameters.	8
8	Pointer to a constant.	9
9	Returning a pointer from a function.	10
10	New demo.	11
11	Delete() in a function.	12
12	Pseudodynamic array.	13
13	Dynamic array example.	14

```
1 // 10-1.cpp -- address operator
2 #include <iostream>
3 using namespace std;
4
5 char letter;
6 short number;
7 float amount;
8 double profit;
9 char ch;
10
11 int main() {
12     cout << "Addressses of variables\n";
13     cout << "Address of letter is: "
14         << long(&letter) << endl;
15     cout << "Address of number is: "
16         << long(&number) << endl;
17     cout << "Address of amount is: "
18         << long(&amount) << endl;
19     cout << "Address of profit is: "
20         << long(&profit) << endl;
21     cout << "Address of ch is: "
22         << long(&ch) << endl;
23
24     return 0;
25 }
```

Figure 1: §10.1 Address operator demo.
Source file: 10-1.cpp

```

1 // pointer demo
2
3 #include<iostream>
4 using namespace std;
5
6 int main() {
7     int x = 25;
8     int* ptr;    // ptr points to an int
9
10    ptr = &x;    // Store address of x in ptr
11    cout << "The value of x is " << x << endl;
12    cout << "The address of x is " << ptr << endl;
13    cout << "*ptr points to a value of " << *ptr << endl;
14    cout << "&x points to address " << &x << endl;
15
16    return 0;
17 }

```

Screen Shot 2016-09-11 at 6.48.34... Open with Preview

```

Calebs-iMac:Desktop calebfowler$ ./a.out
The value of x is 25
The address of x is 0x7fff5f593ba8
*ptr points to a value of 25
&x points to address 0x7fff5f593ba8

```

Figure 2: §10.2 Difference between regular and pointer variables. Source file: pointerDemo.cpp

```

1 // 10-4.cpp -- point to different vars
2 #include <iostream>
3 using namespace std;
4
5 int main()
6 {
7     int x = 25,
8         y = 50,
9         z = 75;
10    int* ptr;
11
12    cout << "Here are the values of x, y, and z\n";
13    cout << x << ", " << y << ", " << z << endl;
14
15    // manipulate with ptr
16    ptr = &x;    // store address
17    *ptr *= 2;    // value times 2
18    ptr = &y;
19    *ptr *= 2;
20
21    ptr = &z;
22    *ptr *= 2;
23
24    cout << "Here are the new values\n";
25    cout << x << ", " << y << ", " << z << endl;
26
27    return 0;
28 }

```

Figure 3: §10.2 Demo access different variables with pointers. Source file: 10-4.cpp

```

//10-5.cpp -- Arrays & Pointers
#include <iostream>
using namespace std;

int main() {
    short numbers[] = {10, 20, 30, 40, 50};

    cout << "The first array element is "
         << *numbers << endl;

    return 0;
}

```

Figure 4: §10.3 Relationship between arrays and pointers. Source file: 10-5.cpp

```

1 // 10-6.cpp -- Use arrays with pointer notation
2 #include <iostream>
3 using namespace std;
4
5 int main()
6 {
7     const int SIZE = 5;
8     int numbers[SIZE];
9
10    cout << "Enter " << SIZE << " numbers: ";
11    for(int count = 0; count < SIZE; count++)
12        cin >> *(numbers + count);
13    cout << numbers << endl;
14
15    cout << "Here are the inputs\n";
16    for (int count = 0; count < SIZE; count++)
17        cout << *(numbers + count) << " ";
18    cout << endl;
19
20    return 0;
21 }

```

Figure 5: §10.3 Arrays with pointer notation. Source file: 10-6.cpp

```

1 //6-26(mod).cpp -- Reference Variable Demo (modified)
2 // from C++ Brief p.351
3 #include <iostream>
4 using namespace std;
5
6 // Function Prototypes
7 void doubleNum(int &);
8 void getNum(int &);
9
10 int main() {
11     int value;
12     getNum(value);
13     doubleNum(value);
14     cout << "Doubling that number is " << value << endl;
15
16     return 0;
17 }
18
19 void getNum(int& userNum) {
20     cout << "Enter a number: ";
21     cin >> userNum;
22 }
23
24 void doubleNum(int& refVar) {
25     refVar *= 2;
26 }

```

Figure 6: §Ref Var Review Demo of reference variables. Source file: 6-26(mod).cpp

```

1 // 10-11.cpp -- Pointers as function parameters
2 #include <iostream>
3 using namespace std;
4
5 void getNumber(int*);
6 void doubleValue(int*);
7
8 int main()
9 {
10     int number;
11     getNumber(&number);
12     doubleValue(&number);
13     cout << "Value doubled is " << number << endl;
14 }
15
16 void getNumber(int* input)
17 {
18     cout << "Enter an integer number: ";
19     cin >> *input;
20 }
21
22 void doubleValue(int* val)
23 {
24     *val *= 2;
25 }
26

```

Figure 7: §10.7 Pointers as function parameters. Source file: 10-11.cpp


```

// 10-13.cpp -- pointer to const demo
#include <iostream>
using namespace std;

void displayValues(const int* numbers, int size);

int main()
{
    const int SIZE = 6;
    const int array[SIZE] = {1, 2, 3, 4, 5, 6};
    int array2[SIZE] = {10, 20, 30, 40, 50, 60};

    displayValues(array, SIZE);
    displayValues(array2, SIZE);

    return 0;
}

void displayValues(const int* numbers, int size) {
    for (int count = 0; count < size; count++) {
        cout << *(numbers + count) << ", ";
    }
    cout << endl;
}

```

Figure 8: §10.8 Pointer to a constant.
Source file: 10-13.cpp

```

1 // 10-15.cpp - returning a pointer from a function
2 #include <iostream>
3 #include <cstdlib> // rand and srand
4 #include <ctime> // time function
5 #include <assert.h>
6 // #define NDBUG
7 using namespace std;
8
9 // Function Prototypes
10 int* getRandomNumbers(int);
11
12 int main()
13 {
14     int* numbers = nullptr; // point to the numbers
15
16     numbers = getRandomNumbers(5); // build the array
17
18     for (int count = 0; count < 5; count++)
19         cout << numbers[count] << endl;
20
21     delete [] numbers;
22     numbers = nullptr;
23     return 0;
24 }
25
26 int* getRandomNumbers(int size)
27 {
28     assert(size > 0); // protect our function
29     int* array = nullptr; // array to hold numbers
30
31     array = new int[size];
32
33     srand(time(0)); // seed with time
34
35     for (int count = 0; count < size; count++)
36         array[count] = rand();
37     return array;
38 }

```

Figure 9: §10.10 Returning a pointer from a function. Source file: 10-15.cpp

```

1 // new.cpp -- introduction to new()
2 #include <iostream>
3 #include <string>
4 using namespace std;
5
6 int* MakeHeap();
7 void ShowHeap(string, int*);
8
9 int main() {
10     int* pInt = nullptr;
11     pInt = MakeHeap();
12
13     ShowHeap("Data in new heap variable: ", pInt);
14
15     *pInt = 5;
16     ShowHeap("Updated data in new heap variable: ", pInt);
17
18     return 0;
19 }
20
21 int* MakeHeap() {
22     int* pTmp = nullptr;
23     pTmp = new int;
24     return pTmp;
25 }
26
27
28 void ShowHeap(string msg, int* pTmp) {
29     cout << msg << *pTmp << endl;
30 }
31 }

```

Figure 10: §10.9 New demo. Source file: new.cpp

```

1 // new2.cpp -- introduction to new()
2 #include <iostream>
3 #include <string>
4 using namespace std;
5
6 int* MakeHeap(int);
7 void ShowHeap(string, int*);
8
9 int main() {
10     int* pInt = nullptr;
11     pInt = MakeHeap(10);
12
13     ShowHeap("Data in new heap variable: ", pInt);
14
15     // *pInt = 5; // now this won't work.
16     delete pInt;
17     pInt = nullptr;
18     pInt = MakeHeap(50);
19
20     ShowHeap("Updated data in new heap variable: ", pInt);
21
22     return 0;
23 }
24
25 // This now returns a pointer to a constant
26 int* MakeHeap(int n) { ...
31 }
32
33
34 void ShowHeap(string msg, int* pTmp) {
35     cout << msg << *pTmp << endl;
36
37 }

```

Figure 11: §10.9 Delete() in a function.
Source file: new2.cpp

```

1 // 10-14.cpp Pseudo Dynamic Memory
2 #include <iostream>
3 #include <iomanip>
4 using namespace std;
5
6 int main(){
7     double* sales = nullptr,    // set to no address
8         total = 0.0,            // accumulator
9         average;                // Average sales
10    int numDays;                // number days sales
11
12    // Get the number of days sales
13    cout << "How many days of sales figures do you wish ";
14    cout << "to process? ";
15    cin >> numDays;
16
17    // Dynamically allocate array - 1 time
18    sales = new double[numDays]; // allocate memory
19
20    // Get sales for each day
21    cout << "Enter the sales figures below.\n";
22    for (int count = 0; count < numDays; count++){
23        cout << "Day " << (count + 1) << ": ";
24        cin >> sales[count];
25    }
26
27    // Calculate total sales
28    for (int count = 0; count < numDays; count++){
29        total += sales[count];
30    }
31    average = total / numDays; // Calculate average sales per day
32
33    // Display
34    cout << setprecision(2) << fixed << showpoint;
35    cout << "\n\nTotal Sales: $" << total << endl;
36    cout << "Average Sales: $" << average << endl;
37
38    // Free Memory!
39    delete[] sales;
40    sales = nullptr;
41    return 0;
42 }

```

Figure 12: §10.9 Pseudodynamic array.
Source file: 10-14.cpp

```

1 // rdaa.cpp -- real dynamically allocated array
2 #include <iostream>
3 using namespace std;
4
5 int main(){ // build the initial array
6     int* pInt = nullptr;
7     int size = 1;
8
9     pInt = new int[size];
10    pInt[0] = 10;
11
12    for (int i = 0; i < size; i++)
13        cout << pInt[i] << " ";
14    cout << endl;
15
16    // increase by 1
17    int* pTemp = nullptr;
18    size++;
19    pTemp = new int[size];
20
21    pTemp[0] = pInt[0];
22    pTemp[1] = 20;
23
24    delete[] pInt;
25    pInt = pTemp;
26    pTemp = nullptr;
27
28    for (int i = 0; i < size; i++)
29        cout << pInt[i] << " ";
30    cout << endl;
31
32    // decrease by 1 from the right
33    // int* pTemp = nullptr;
34    size--;
35    pTemp = new int[size];
36    pTemp[0] = pInt[0];
37
38    delete[] pInt;
39    pInt = pTemp;
40    pTemp = nullptr;
41
42    for (int i = 0; i < size; i++)
43        cout << pInt[i] << " ";
44    cout << endl;
45
46    return 0;
47 }

```

Figure 13: §10.9 Dynamic array example. Source file: rdaa.cpp