Chapter 10 - Pointers

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.*

Idea	Notes

List of Figures

Demonstration Code.

```cpp
// 10-1.cpp -- address operator
#include <iostream>
using namespace std;

char letter;
short number;
float amount;
double profit;
char ch;

int main() {
  cout << "Addressses of variables\n";
  cout << "Address of letter is: "
    << long(&letter) << endl;
  cout << "Address of number is: "
    << long(&number) << endl;
  cout << "Address of amount is: "
    << long(&amount) << endl;
  cout << "Address of profit is: "
    << long(&profit) << endl;
  cout << "Address of ch is: "
    << long(&ch) << endl;

  return 0;
}
```

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

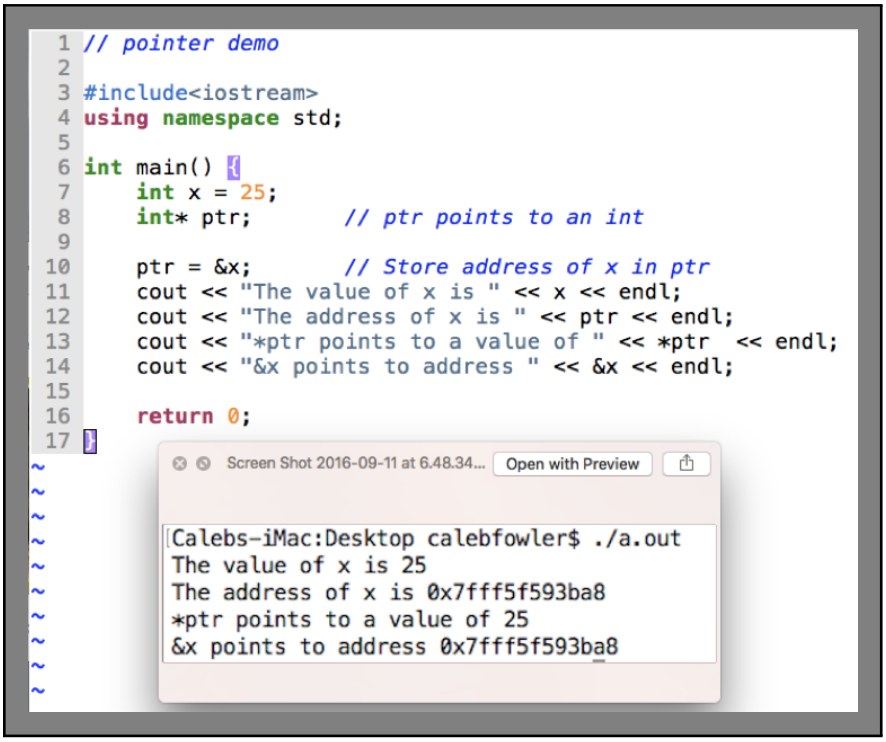


Figure 2: §10.2 Difference between regular and pointer variables. Source file: pointerDemo.cpp

```cpp
// 10-4.cpp -- point to different vars
#include <iostream>
using namespace std;

int main()
{
  int x = 25,
      y = 50,
      z = 75;
  int* ptr;

  cout << "Here are the values of x, y, and z\n";
  cout << x << ", " << y << ", " << z << endl;

  // manipulate with ptr
  ptr = &x;     // store address
  *ptr *= 2;    // value times 2
  ptr = &y;
  *ptr *= 2;

  ptr = &z;
  *ptr *= 2;

  cout << "Here are the new values\n";
  cout << x << ", " << y << ", " << z << endl;

  return 0;
}
```

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

```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;
}
```

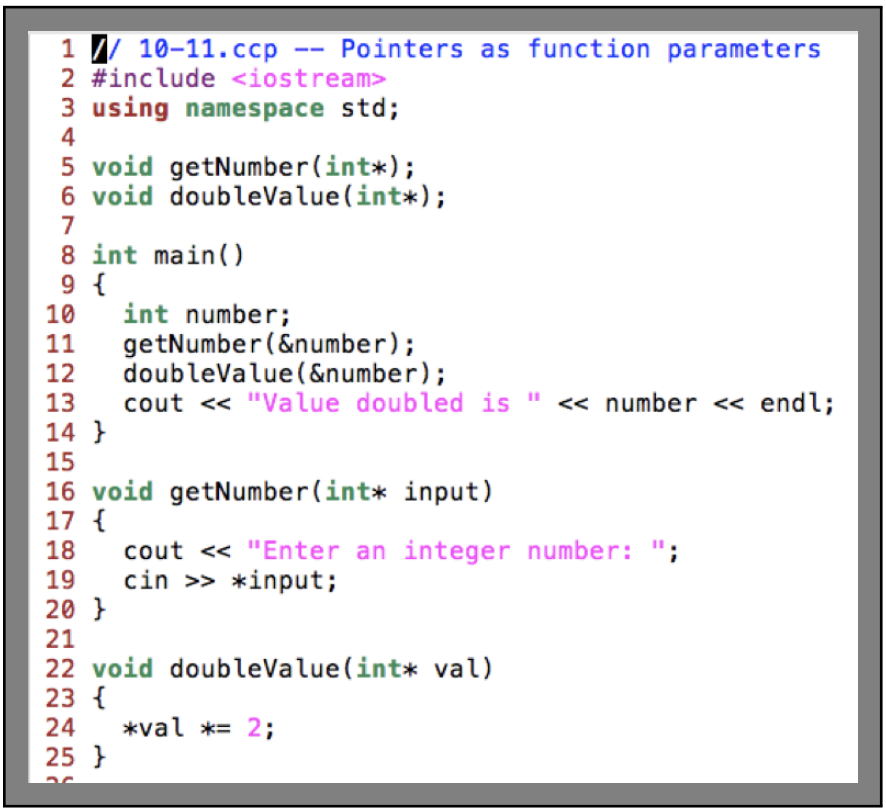
Figure 4: §10.3 Relationship between arrays and pointers. Source file: 10-5.cpp

```cpp
// 10-6.cpp -- Use arrays with pointer notation
#include <iostream>
using namespace std;

int main()
{
  const int SIZE = 5;
  int numbers[SIZE];

  cout << "Enter " << SIZE << " numbers: ";
  for(int count = 0; count < SIZE; count++)
    cin >> *(numbers + count);
  cout << numbers << endl;

  cout << "Here are the inputs\n";
  for (int count = 0; count < SIZE; count++)
    cout << *(numbers + count) << " ";
  cout << endl;

  return 0;
}
```

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

```cpp
//6-26(mod).cpp -- Reference Variable Demo (modified)
// from C++ Brief p.351
#include <iostream>
using namespace std;

// Function Prototypes
void doubleNum(int &);
void getNum(int &);

int main() {
  int value;
  getNum(value);
  doubleNum(value);
  cout << "Doubling that number is " << value << endl;

  return 0;
}

void getNum(int& userNum) {
  cout << "Enter a number: ";
  cin >> userNum;
}

void doubleNum(int& refVar) {
  refVar *= 2;
}
```

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



```
 1 // 10-11.ccp -- 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 }
```

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

```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;
}
```

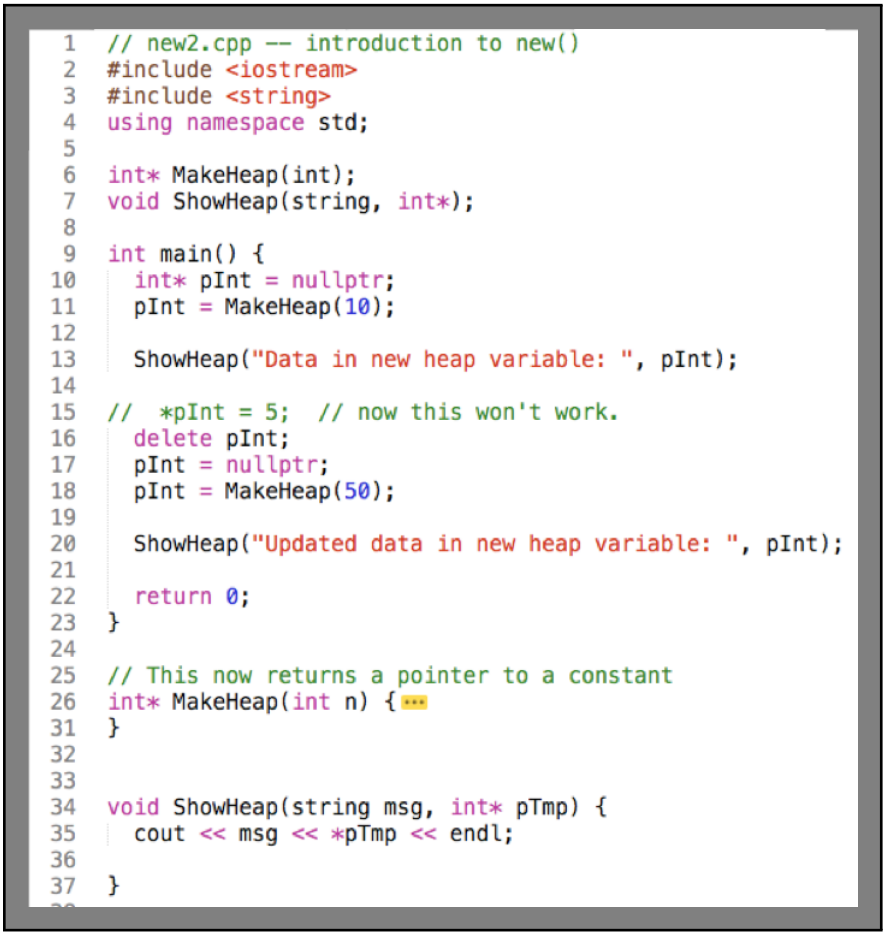
Figure 8: §10.8 Pointer to a constant.
Source file: 10-13.cpp

```cpp
// 10-15.cpp - returning a pointer from a function
#include <iostream>
#include <cstdlib>      // rand and srand
#include <ctime>        // time function
#include <assert.h>
//#define NDBUG
using namespace std;

// Function Prototypes
int* getRandomNumbers(int);

int main()
{
    int* numbers = nullptr; // point to the numbers

    numbers = getRandomNumbers(5);  // build the array

    for (int count = 0; count < 5; count++)
    cout << numbers[count] << endl;

    delete [] numbers;
    numbers = nullptr;
    return 0;
}

int* getRandomNumbers(int size)
{
    assert(size > 0);       // protect our function
    int* array = nullptr;   // array to hold numbers

    array = new int[size];

    srand(time(0));     // seed with time

    for (int count = 0; count < size; count++)
    array[count] = rand();
    return array;
}
```

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

```cpp
// new.cpp -- introduction to new()
#include <iostream>
#include <string>
using namespace std;

int* MakeHeap();
void ShowHeap(string, int*);

int main() {
  int* pInt = nullptr;
  pInt = MakeHeap();

  ShowHeap("Data in new heap variable: ", pInt);

  *pInt = 5;
  ShowHeap("Updated data in new heap variable: ", pInt);

  return 0;
}

int* MakeHeap() {
  int* pTmp = nullptr;
  pTmp = new int;
  return pTmp;
}


void ShowHeap(string msg, int* pTmp) {
  cout << msg << *pTmp << endl;

}
```

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



```cpp
// new2.cpp -- introduction to new()
#include <iostream>
#include <string>
using namespace std;

int* MakeHeap(int);
void ShowHeap(string, int*);

int main() {
  int* pInt = nullptr;
  pInt = MakeHeap(10);

  ShowHeap("Data in new heap variable: ", pInt);

//  *pInt = 5;  // now this won't work.
  delete pInt;
  pInt = nullptr;
  pInt = MakeHeap(50);

  ShowHeap("Updated data in new heap variable: ", pInt);

  return 0;
}

// This now returns a pointer to a constant
int* MakeHeap(int n) {...
}


void ShowHeap(string msg, int* pTmp) {
  cout << msg << *pTmp << endl;

}
```

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

```cpp
// 10-14.cpp Pseudo Dynamic Memory
#include <iostream>
#include <iomanip>
using namespace std;

int main(){
    double* sales = nullptr,    // set to no address
        total = 0.0,              // accumulator
        average;                    // Average sales
    int numDays;                  // number days sales

    // Get the number of days sales
    cout << "How many days of sales figures do you wish ";
    cout << "to process? ";
    cin >> numDays;

    // Dynamically allocate array - 1 time
    sales = new double[numDays];    // allocate memory

    // Get sales for each day
    cout << "Enter the sales figures below.\n";
    for (int count = 0; count < numDays; count++){
        cout << "Day " << (count + 1) << ": ";
        cin >> sales[count];
    }

    // Calculate total sales
    for (int count = 0; count < numDays; count++)
        total += sales[count];
    average = total / numDays;  // Calculate average sales per day

    // Display
    cout << setprecision(2) << fixed << showpoint;
    cout << "\n\nTotal Sales: $" << total << endl;
    cout << "Average Sales: $" << average << endl;

    // Free Memory!
    delete[] sales;
    sales = nullptr;
    return 0;
}
```

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

```cpp
// rdaa.cpp  -- real dynamically allocated array
#include <iostream>
using namespace std;

int main(){     // build the initial array
    int* pInt = nullptr;
    int size = 1;

    pInt = new int[size];
    pInt[0] = 10;

    for (int i = 0; i < size; i++)
    cout << pInt[i] << "  ";
    cout << endl;

// increase by 1
    int* pTemp = nullptr;
    size++;
    pTemp = new int[size];

    pTemp[0] = pInt[0];
    pTemp[1] = 20;

    delete[] pInt;
    pInt = pTemp;
    pTemp = nullptr;

    for (int i = 0; i < size; i++)
    cout << pInt[i] << "  ";
    cout << endl;

    // decrease by 1 from the right
//    int* pTemp = nullptr;
    size--;
    pTemp = new int[size];
    pTemp[0] = pInt[0];

    delete[] pInt;
    pInt = pTemp;
    pTemp = nullptr;

    for (int i = 0; i < size; i++)
    cout << pInt[i] << "  ";
    cout << endl;

    return 0;
}
```

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