Most of the contents are taken from Tony Gaddies slides

Eliyas Girma

CSC 322 Lab Tutorial 1

Introduction

- We have started a revision about pointers. Here we continue from where we stopped.
- The main topics here we cover are
  - Pointer as a function parameter
  - Returning pointer from a function
  - How to allocate dynamic memory

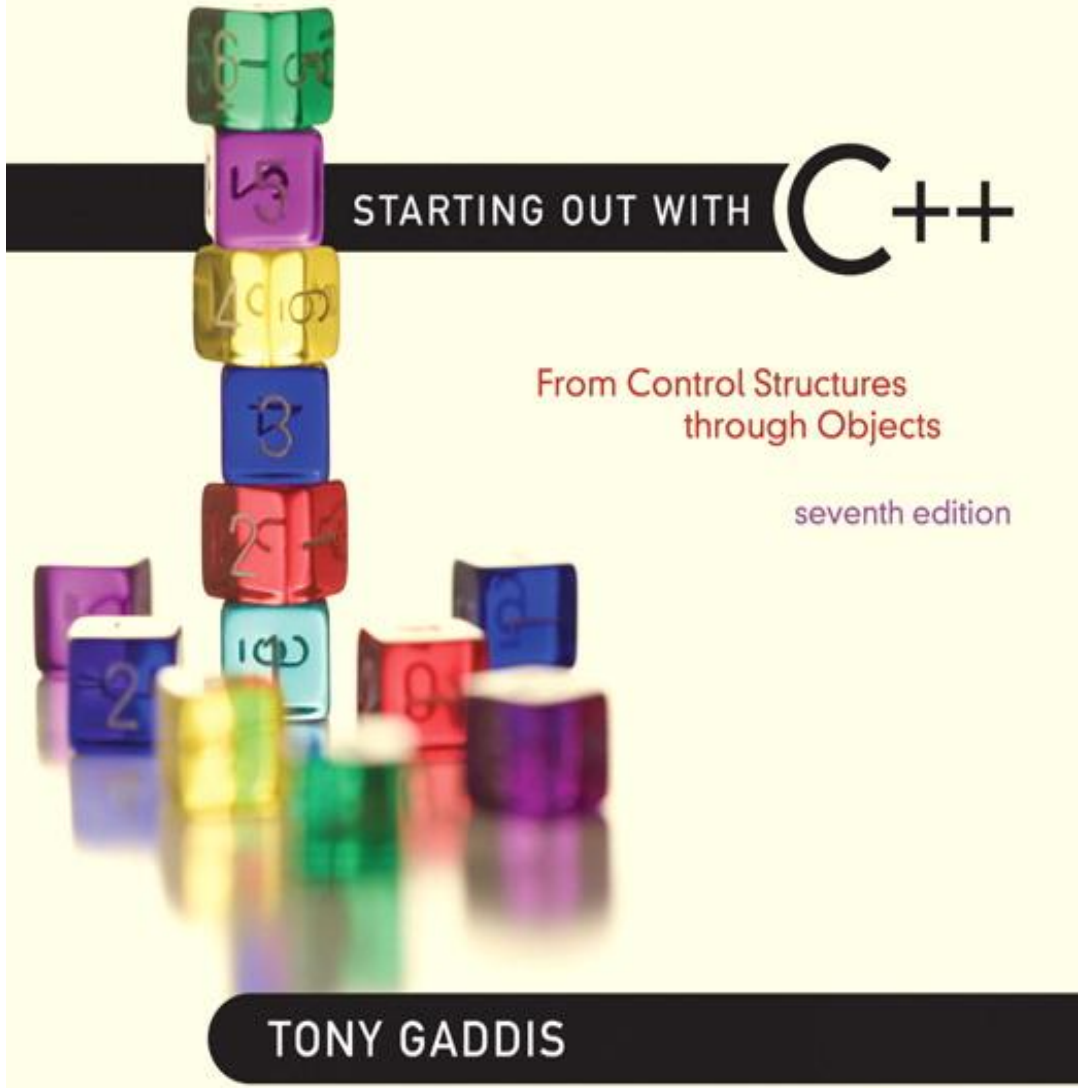
Introduction..

- After finishing this slide you should understand how a dynamic memory is allocated using pointers and you are supposed to comment the previous discussed matrix manipulation example we had in the class.
- The codes for the matrix manipulation examples are selected and attached at the end of the slide.
- Deadline for submission for the commented code is March 25. Include Name, ID and Section. It is individual assignment. Submit for representatives

Chapter 7:

Lecture 11

Pointers



Addison-Wesley
is an imprint of
PEARSON

7.4

Pointer Arithmetic

Pointer Arithmetic

- Operations on pointer variables:

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

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

`+, -` (pointer and `int`)	`cout << *(valptr + 2); // 11`

`+=, -=` (pointer and `int`)	`valptr = vals; // points at 4`
	`valptr += 2;  // points at 11`

`-` (pointer from pointer)	`cout << valptr-val; // difference`
	`//(number of ints) between valptr`
	`// and val`

From Program 7-9

```
7     const int SIZE = 8;
8     int set[SIZE] = {5, 10, 15, 20, 25, 30, 35, 40};
9     int *numPtr;   // Pointer
10    int count;     // Counter variable for loops
11
12    // Make numPtr point to the set array.
13    numPtr = set;
14
15    // Use the pointer to display the array contents.
16    cout << "The numbers in set are:\n";
17    for (count = 0; count < SIZE; count++)
18    {
19       cout << *numPtr << " ";
20       numPtr++;
21    }
22
23    // Display the array contents in reverse order.
24    cout << "\nThe numbers in set backward are:\n";
25    for (count = 0; count < SIZE; count++)
26    {
27       numPtr--;
28       cout << *numPtr << " ";
29    }
```

Program Output

```
The numbers in set are:
5 10 15 20 25 30 35 40
The numbers in set backward are:
40 35 30 25 20 15 10 5
```

7.5

Initializing Pointers

Initializing Pointers

- Can initialize at definition time:

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

- Cannot mix data types:

```
double cost;
int *ptr = &cost; // won't work
```

7.6

Comparing Pointers

Comparing Pointers

- Relational operators (<, >=, etc.) can be used to compare addresses in pointers
- Comparing addresses in pointers is not the same as comparing contents pointed at by pointers:

```
if (ptr1 == ptr2)    // compares
                     // addresses
if (*ptr1 == *ptr2)  // compares
                     // contents
```

9.7

Pointers as Function Parameters

Pointers as Function Parameters

- A pointer can be a parameter
- Works like reference variable to allow change to argument from within function
- Requires:

1) asterisk * on parameter in prototype and heading

```
void getNum(int *ptr); // ptr is pointer to an int
```

2) asterisk ***** in body to dereference the pointer

```
cin >> *ptr;
```

3) address as argument to the function

```
getNum(&num);      // pass address of num to getNum
```

Example

```
void swap(int *x, int *y)
{     int temp;
      temp = *x;
      *x = *y;
      *y = temp;
}

int num1 = 2, num2 = -3;
swap(&num1, &num2);
```

Program 9-11

```
1  // This program uses two functions that accept addresses of
2  // variables as arguments.
3  #include <iostream>
4  using namespace std;
5
6  // Function prototypes
7  void getNumber(int *);
8  void doubleValue(int *);
9
10 int main()
11 {
12    int number;
13
14    // Call getNumber and pass the address of number.
15    getNumber(&number);
16
17    // Call doubleValue and pass the address of number.
18    doubleValue(&number);
19
20    // Display the value in number.
21    cout << "That value doubled is " << number << endl;
22    return 0;
23 }
24
```

(Program Continues)

Program 9-11 *(continued)*

```
25  //*************************************************************
26  // Definition of getNumber. The parameter, input, is a pointer. *
27  // This function asks the user for a number. The value entered  *
28  // is stored in the variable pointed to by input.               *
29  //*************************************************************
30
31  void getNumber(int *input)
32  {
33     cout << "Enter an integer number: ";
34     cin >> *input;
35  }
36
37  //*************************************************************
38  // Definition of doubleValue. The parameter, val, is a pointer. *
39  // This function multiplies the variable pointed to by val by   *
40  // two.                                                         *
41  //*************************************************************
42
43  void doubleValue(int *val)
44  {
45     *val *= 2;
46  }
```

Program Output with Example Input Shown in Bold

```
Enter an integer number: 10 [Enter]
That value doubled is 20
```

Pointers to Constants

- If we want to store the address of a constant in a pointer, then we need to store it in a pointer-to-const.

Pointers to Constants

- Example: Suppose we have the following definitions:

```
const int SIZE = 6;
const double payRates[SIZE] =
     { 18.55, 17.45, 12.85,
       14.97, 10.35, 18.89 };
```

- In this code, `payRates` is an array of constant doubles.

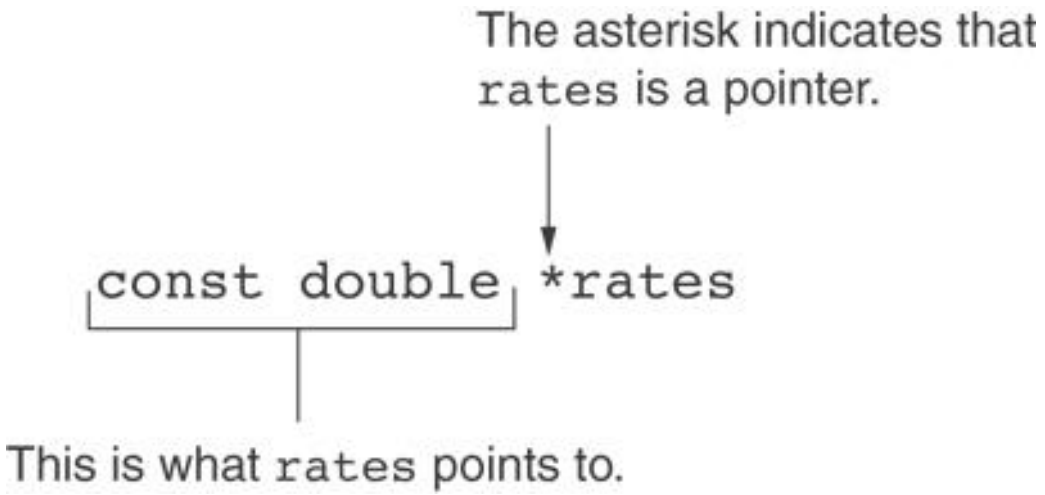
Pointers to Constants

- Suppose we wish to pass the `payRates` array to a function? Here's an example of how we can do it.

```
void displayPayRates(const double *rates, int size)
{
   for (int count = 0; count < size; count++)
   {
      cout << "Pay rate for employee " << (count + 1)
           << " is $" << *(rates + count) << endl;
   }
}
```

The parameter, rates, is a pointer to `const double`.

Declaration of a Pointer to Constant



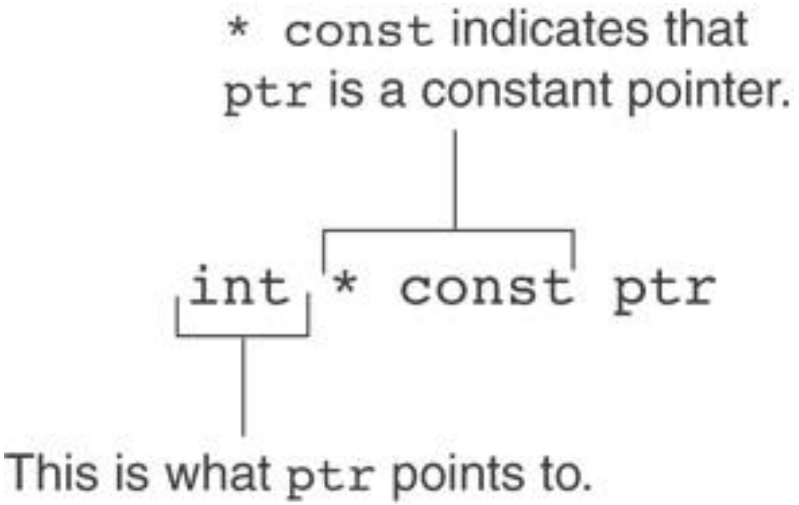
Constant Pointers

- A constant pointer is a pointer that is initialized with an address, and cannot point to anything else.

- Example

```
int value = 22;
int * const ptr = &value;
```

Constant Pointers



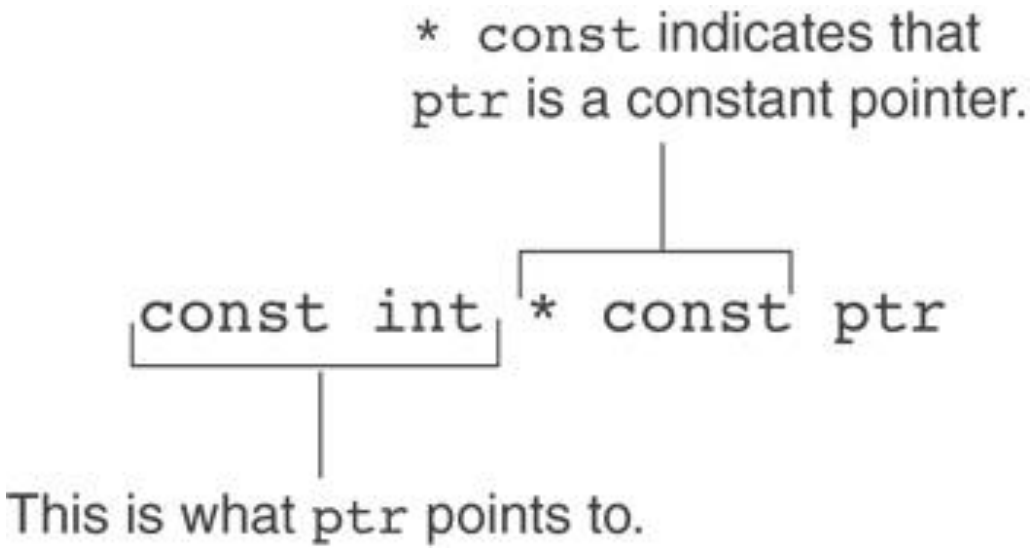
Constant Pointers to Constants

- A constant pointer to a constant is:
  - a pointer that points to a constant
  - a pointer that cannot point to anything except what it is pointing to

- Example:

```
int value = 22;
const int * const ptr = &value;
```

Constant Pointers to Constants



7.8

Dynamic Memory Allocation

Dynamic Memory Allocation

- Can allocate storage for a variable while program is running
- Computer returns address of newly allocated variable
- Uses `new` operator to allocate memory:

```
double *dptr;
dptr = new double;
```

- `new` returns address of memory location

Dynamic Memory Allocation

- Can also use `new` to allocate array:

```
const int SIZE = 25;
arrayPtr = new double[SIZE];
```

- Can then use `[]` or pointer arithmetic to access array:

```
for(i = 0; i < SIZE; i++)
   arrayptr[i] = i * i;
```

or

```
for(i = 0; i < SIZE; i++)
   *(arrayptr + i) = i * i;
```

- Program will terminate if not enough memory available to allocate

Releasing Dynamic Memory

- Use `delete` to free dynamic memory:

```
delete fptr;
```

- Use `[]` to free dynamic array:

```
delete [] arrayptr;
```

- Only use `delete` with dynamic memory!

Program 9-14

```
1  // This program totals and averages the sales figures for any
2  // number of days. The figures are stored in a dynamically
3  // allocated array.
4  #include <iostream>
5  #include <iomanip>
6  using namespace std;
7
8  int main()
9  {
10    double *sales,       // To dynamically allocate an array
11           total = 0.0, // Accumulator
12           average;     // To hold average sales
```

Program 9-14 *(continued)*

```
13      int numDays,         // To hold the number of days of sales
14          count;           // Counter variable
15
16      // Get the number of days of sales.
17      cout << "How many days of sales figures do you wish ";
18      cout << "to process? ";
19      cin >> numDays;
20
21      // Dynamically allocate an array large enough to hold
22      // that many days of sales amounts.
23      sales = new double[numDays];
24
25      // Get the sales figures for each day.
26      cout << "Enter the sales figures below.\n";
27      for (count = 0; count < numDays; count++)
28      {
29         cout << "Day " << (count + 1) << ": ";
30         cin >> sales[count];
31      }
32
```

Program 9-14 (Continued)

```
33      // Calculate the total sales
34      for (count = 0; count < numDays; count++)
35      {
36         total += sales[count];
37      }
38
39      // Calculate the average sales per day
40      average = total / numDays;
41
42      // Display the results
43      cout << fixed << showpoint << setprecision(2);
44      cout << "\n\nTotal Sales: $" << total << endl;
45      cout << "Average Sales: $" << average << endl;
46
47      // Free dynamically allocated memory
48      delete [] sales;
49      sales = 0;        // Make sales point to null.
50
51      return 0;
52  }
```

Program Output with Example Input Shown in Bold

```
How many days of sales figures do you wish to process? 5 [Enter]
Enter the sales figures below.
Day 1: 898.63 [Enter]
Day 2: 652.32 [Enter]
Day 3: 741.85 [Enter]
Day 4: 852.96 [Enter]
Day 5: 921.37 [Enter]

Total Sales: $4067.13
Average Sales: $813.43
```

Notice that in line 49 the value 0 is assigned to the `sales` pointer. It is a good practice to store 0 in a pointer variable after using delete on it. First, it prevents code from inadvertently using the pointer to access the area of memory that was freed. Second, it prevents errors from occurring if `delete` is accidentally called on the pointer again. The `delete` operator is designed to have no effect when used on a null pointer.

7.9

Returning Pointers from Functions

Returning Pointers from Functions

- Pointer can be the return type of a function:

```
int* newNum();
```

- The function must not return a pointer to a local variable in the function.
- A function should only return a pointer:
  - to data that was passed to the function as an argument, or
  - to dynamically allocated memory

From Program 7-15

```
34  int *getRandomNumbers(int num)
35  {
36     int *array;    // Array to hold the numbers
37
38     // Return null if num is zero or negative.
39     if (num <= 0)
40        return NULL;
41
42     // Dynamically allocate the array.
43     array = new int[num];
44
45     // Seed the random number generator by passing
46     // the return value of time(0) to srand.
47     srand( time(0) );
48
49     // Populate the array with random numbers.
50     for (int count = 0; count < num; count++)
51        array[count] = rand();
52
53     // Return a pointer to the array.
54     return array;
55  }
```

Matrix manipulation exercise

- You should comment each line with what it is and it does.
- You need to give name for each functions that are labeled from f1 to f6

//Function prototypes

```
int** f1();
void f2(int** arr1,int** arr2);
void f3(int** arr1,int** arr2);
int** f4(int** arr1,int** arr2);
void f5(int** arr);
void f6(int** arr);

const int col= 6,row=6;// These values can be given with the users preference.
```

You should comment each line

```
int** f1()
{
   int** arr = new int*[col];
   for (int i=0; i<3;i++)
      {
         arr[i]=new int[row];
      for(int j=0;j<3;j++)
         {
            cin>> arr[i][j];

         }
         }

return arr;
}
```

```
void f2(int** arr1,int** arr2)
{

   cout<<“I did *********\n";
  int c[row][col];

    for (int i=0; i<row;i++)
      {
      for(int j=0;j<col;j++)
         {
            c[i][j]=arr1[i][j]+ arr2[i][j];
          cout<< c[i][j]<<" ";
         }
         cout<<"\n";
      }

}
```

```
void f3(int** arr1,int** arr2)
{

    cout<<“I did *********\n";
  int c[row][col];

    for (int i=0; i<row;i++)
      {
      for(int j=0;j<col;j++)
         {
            c[i][j]=arr1[i][j] - arr2[i][j];
          cout<< c[i][j]<<" ";
         }
         cout<<"\n";
      }

}
```

```
void f6(int** arr)
{
for (int i=0; i<row;i++)
        {
        for(int j=0;j<col;j++)
           {
            cout<< (arr[j][i])<<" ";
           }
           cout<<"\n";
        }

         cout<<"************ \n";
}
```

```
int** f4(int** arr1,int** arr2)
{
    int** c = new int*[col];

      for (int i=0; i<col;i++)
      {
         c[i]=new int[row];

      for(int j=0;j<col;j++)
         {
             for(int k=0;k<row;k++)
         {
          c[i][j] +=arr1[i][k] * arr2[k][j];
         }
      }
      }
return c;
}
```

```
void f5(int** arr)
{
   cout<<"********** \n";
   for (int i=0; i<row;i++)
      {
      for(int j=0;j<col;j++)
         {
          cout<< (arr[i][j])<<" ";
         }
         cout<<"\n";
      }

       cout<<"********** \n";
}
```