Arrays and Strings

Prof. Zankhana Dabhi
Department of Information Technology,
D. D. University, Nadiad.

Arrays

- An array in C or C++ is a collection of items stored at contiguous memory locations.
- An elements of an array can be accessed randomly using indices of an array.
- They can be used to store collection of primitive data types as well as user defined data types.
- It is used to represent many instances in one variable.

Array Declaration

- Different ways of declaring array:

  1. int a[10]; OR int n = 10; int arr2[n];

  2. int arr[] = { 10, 20, 30, 40 } ;

  3. int arr[6] = { 10, 20, 30, 40 };

Advantages / Disadvantages of Array

- Advantages of an Array:
  - Random access
  - Easy traversal

- Disadvantages of an Array:
  - It is not dynamic
  - Insertion and deletion can be costly

- Accessing Array Elements:

  Array elements are accessed by using an integer index. Array index starts with 0 and goes till size of array minus 1.

- No Index Out of bound Checking:

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

int main()
{
   int arr[2];

   cout << arr[3] << " ";
   cout << arr[-2] << " ";

   return 0;
}
```

```
#include <stdio.h>
int main()
{
    int arr[2] = { 10, 20, 30, 40, 50 };
    return 0;
}
```

- In c, it is not compiler error, it shows just warning.
- In c++, it gives compiler error : ”too many initializers”

Multidimensional Array

- It is an array of array.
- Data in multidimensional arrays are stored in tabular form (in row major order).

Two dimensional array:

```
int two_d[10][20];
```

Three dimensional array:

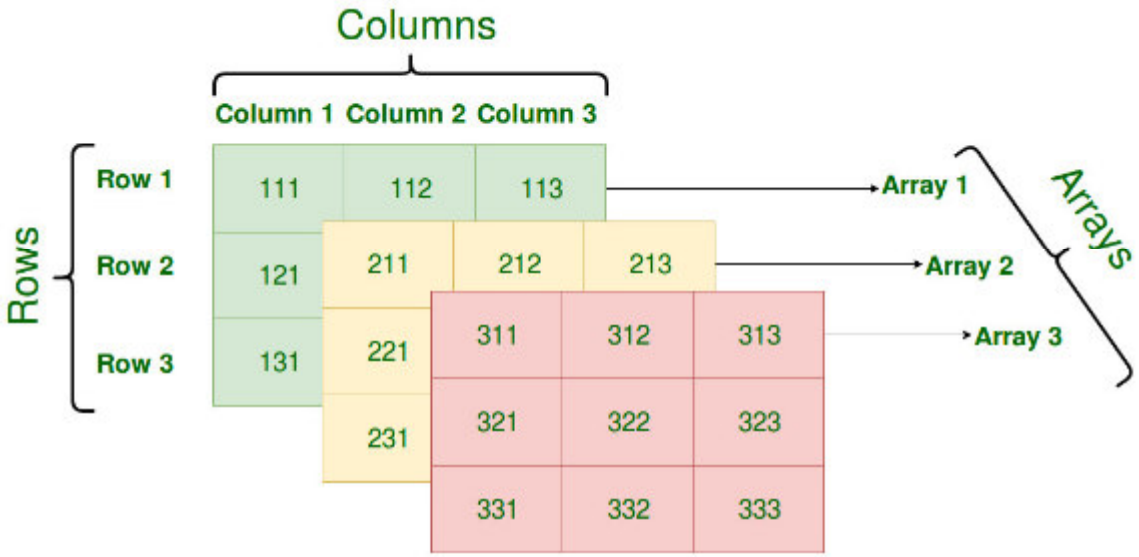
```
int three_d[10][20][30];
```

Initializing Two Dim. Array

	Column 0	Column 1	Column 2
Row 0	x[0][0]	x[0][1]	x[0][2]
Row 1	x[1][0]	x[1][1]	x[1][2]
Row 2	x[2][0]	x[2][1]	x[2][2]

- First Method
  - int x[3][4] = {0, 1 ,2 ,3 ,4 , 5 , 6 , 7 , 8 , 9 , 10 , 11};
  - The elements will be filled in the array in the order, first 4 elements from the left in first row, next 4 elements in second row and so on.
- Second (Better) Method
  - int x[3][4] = {{0,1,2,3}, {4,5,6,7}, {8,9,10,11}};
  - Each set of inner braces represents one row. There are total three rows so there are three sets of inner braces.

Three Dim. Array



Initializing 3-Dim. Array

- First Method
  - int x[2][3][4] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23};
- Second (Better) Method
  - int x[2][3][4] = { { {0,1,2,3}, {4,5,6,7}, {8,9,10,11} }, { {12,13,14,15}, {16,17,18,19}, {20,21,22,23} } };

– Accessing Elements of 3-Dimension Array

  - x[1][2][3]

Array of Structures

```
struct part
{
    int partnumber;
    float cost;
};
```

- Arrays can contain structures as well as simple data types.
- The array of structures is defined in the statement
  - part apart[SIZE];
- Accessing a data item that is a member of a structure that is itself an element of an array
  - apart[n].modelnumber

```
struct part
{
    int partnumber;
    float cost;
};
```

```
int main()    {
    int n;
    part apart[SIZE]; //define array of structures
    for(n=0; n<SIZE; n++) //get values for all members
    {
            cout << endl;
            cout << "Enter part number: ";
            cin >> apart[n].partnumber; //get part number
            cout << "Enter cost: ";
            cin >> apart[n].cost; //get cost
    }
    cout << endl;
    for(n=0; n<SIZE; n++) //show values for all members
    {
            cout << " Part " << apart[n].partnumber;
            cout << " Cost " << apart[n].cost << endl;
    }
    return 0;
}
```

Array as Class Member

- Arrays can be used as data items in classes.
- Example : Implementing Stack Class

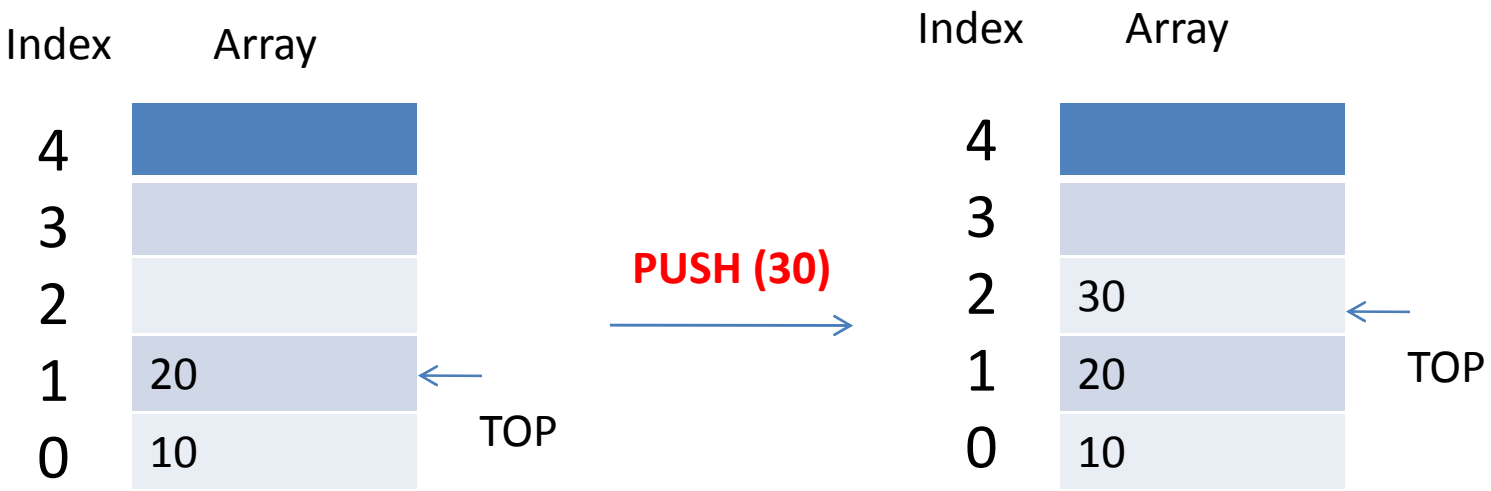
Stack

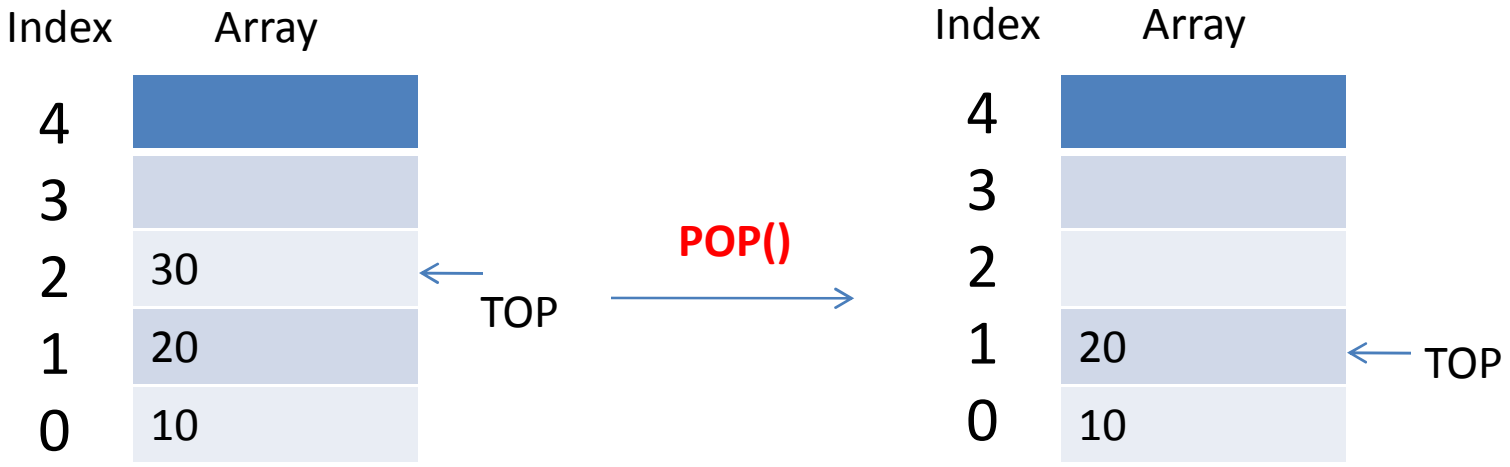
- A stack is a linear data structure that follows the LIFO rule (Last In First Out).
- In a stack, both insertion and deletion take place from just one end, that is, from the top.
- Array implementation of stacks allows the static memory allocation of its data elements

Example : Stack

- A top (int variable) indicates the index of the last item placed on the stack.
- To place an item on the stack—a process called pushing the item—call the push() member function with the value to be stored as an argument.
- To retrieve (or pop) an item from the stack, use the pop() member function, which returns the value of the item.

Stack : Push() and Pop()





```
class Stack     {
private:
      enum { MAX = 10 };
      int st[MAX]; //stack: array of integers
      int top;
public:
      Stack()        //constructor
      { top = 0; }

      void push(int var) //put number on stack
      {
      st[++top] = var;
      }

      int pop() //take number off stack
      {
               return st[top--];
       }
};
```

```
int main()     {

      Stack s1;
      s1.push(11);
      s1.push(22);
      cout << "1: " << s1.pop() << endl; //22
      cout << "2: " << s1.pop() << endl; //11
      s1.push(33);
      s1.push(44);
      s1.push(55);
      s1.push(66);
      cout << "3: " << s1.pop() << endl; //66
      cout << "4: " << s1.pop() << endl; //55
      cout << "5: " << s1.pop() << endl; //44
      cout << "6: " << s1.pop() << endl; //33
      return 0;
}
```

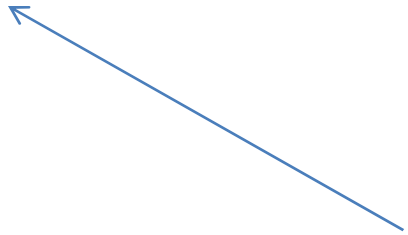
Array of Objects : Example

```
class Distance                  //English Distance class
{
    private:
      int feet;
      float inches;
    public:
      void getdist()  {          //get length from user
            cout << "\n   Enter feet: ";  cin >> feet;
            cout << "   Enter inches: ";  cin >> inches;
    }
    void showdist() {     //display distance
             cout << feet << "\'-" << inches << '\"';
    }
};
```

Array of Objects : Example

```
int main()    {
      Distance dist[100];
      int n=0;
      char ans;
      cout << endl;
      do     {
                cout << "Enter distance number " << n+1;
                dist[n++].getdist();
                cout << "Enter another (y/n)?: ";
                cin >> ans;
      }  while( ans != 'n' );

      for(int j=0; j<n; j++)    {
                cout << "\nDistance number " << j+1 << " is ";
                dist[j].showist();
      }
      cout << endl;
return 0;
}
```



A member function of an object that is an array element is accessed using the dot operator: The array name followed by the index in brackets is joined, using the dot operator, to the member function name followed by parentheses.

Bound-Checking in Arrays

- There is no bounds-checking in C++ arrays.
  - If the program inserts something beyond the end of the array, neither the compiler nor the runtime system will object.
  - However, the new data will be written on top of other data or the program code itself. This may crash the system completely.

Bound-Checking in Arrays

If there is a chance that the user will insert too much data for an array to hold, the array should be made larger or some means of warning should be given to the user.

For example,

```
if( n >= MAX ) {
    cout << “\nThe array is full!!!”;
    break;
}
```

C - Strings

- Two kinds of Strings are commonly used in C++
  - C-strings
  - Strings that are objects of the string class.
- C-strings are arrays of type char.
- They may also be called char* strings, because they can be represented as pointers to type char.
- C-Strings are used in many C library functions
- C-Strings appear in legacy code for years to come

C – String Variable

```
int main()    {
   const int MAX = 80;
   char str[MAX];
   cout << “Enter a string: “;
   cin >> str;
   cout << “You entered: “ << str << endl;
   return 0;
}
```

C – String Variable

- C-strings must terminate with a byte containing 0. This is often represented by the character constant '\0', which is a character with an ASCII value of 0.
- This terminating zero is called the null character.
- When the << operator displays the string, it displays characters until it encounters the null character.

C – String Variable

```
int main()    {
   const int MAX = 80;
   char str[MAX];
   cout << “Enter a string: “;
   cin >> setw(MAX) >> str;
   cout << “You entered: “ << str << endl;
   return 0;
}
```

C – String Variable

- What happens if the user enters a string that is longer than the array used to hold it?
  - There is no built-in mechanism in C++ to keep a program from inserting array elements outside an array.

- The program uses the setw manipulator to specify the maximum number of characters the input buffer can accept.
  - The user may type more characters, but the >> operator won't insert them into the array.
  - One character fewer than the number specified is inserted, so there is room in the buffer for the terminating null character.

String Initialization

```
int main()    {
    char str[] = “Farewell! Hello”;
    cout << str << endl;
    return 0;
}
```

Another approach to initialize the String is

```
char str[] = { ‘F’, ‘a’, ‘r’, ‘e’, ‘w’, ‘e’, ‘l’, ‘l’, ‘!’, ‘H’, ‘e’, ‘l’, ‘l’, ’o’}
```

The last character is a null (zero).

Reading Strings with Blanks

- The extraction operator >> considers a space to be a terminating character for string.
  - It will read strings consisting of a single word, but anything typed after a space is thrown away.
- To read text containing blanks use function, cin.get().
  - get() is a member function of the stream class of which cin is an object.

Reading Strings with Blanks

```
int main()    {
   const int MAX = 80;
   char str[MAX];
   cout << "\nEnter a string: ";
   cin.get(str, MAX);
   cout << "You entered: " << str << endl;
   return 0;
}
```

The first argument to cin::get() is the array address where the string being input will be placed. The second argument specifies the maximum size of the array.

Reading Strings with Multiple Lines

- How to read Strings with Multiple Lines
  - cin::get() function can take a third argument
  - The third argument specifies the character that tells the function to stop reading.
  - The default value for this argument is the newline ('\n') character, but some other character can be specified as this argument, the default will be overridden by the specified character.

Reading Strings with Multiple Lines

```
const int MAX = 2000;
char str[MAX];
int main()    {
    cout << "\nEnter a string:\n";
    cin.get(str, MAX, '$');
    cout << "You entered:\n" << str << endl;
    return 0;
}
```

The function will continue to accept characters until you enter the terminating character (or until the size of the array exceeded). Remember, you must still press Enter after typing the '$' character

Copying String

```
int main()    {
    char str1[] = “Oh, Captain, my Captain! “;
    const int MAX = 80;
    char str2[MAX];
    for(int j=0; j<strlen(str1); j++)
          str2[j] = str1[j];
    str2[j] = ‘\0’;
    cout << str2 << endl;
    return 0;
}
```

Copying String

- The program uses C-string library functions – strlen() - finds the length of a C – string.
- To use string function, the header file CSTRING (or STRING.H) must be included (with include) in the program
  - #include <cstring>
- The string length returned by strlen() does not include the null.
- The copied version of the string must be terminated with a null.
  - If you don't insert this character, the string printed by the program includes all garbage characters following the string
  - The << just keeps on printing characters, whatever they are, until by chance it encounters a '\0'.

Revised Program for Copying String

```
int main()    {
   char str1[] = “Hello World”;
   const int MAX = 80;
   char str2[MAX];
   strcpy(str2, str1);
   cout << str2 << endl;
   return 0;
}
```

- strcpy(destination, source)
  - The variable on the right is copied to the variable on the left.

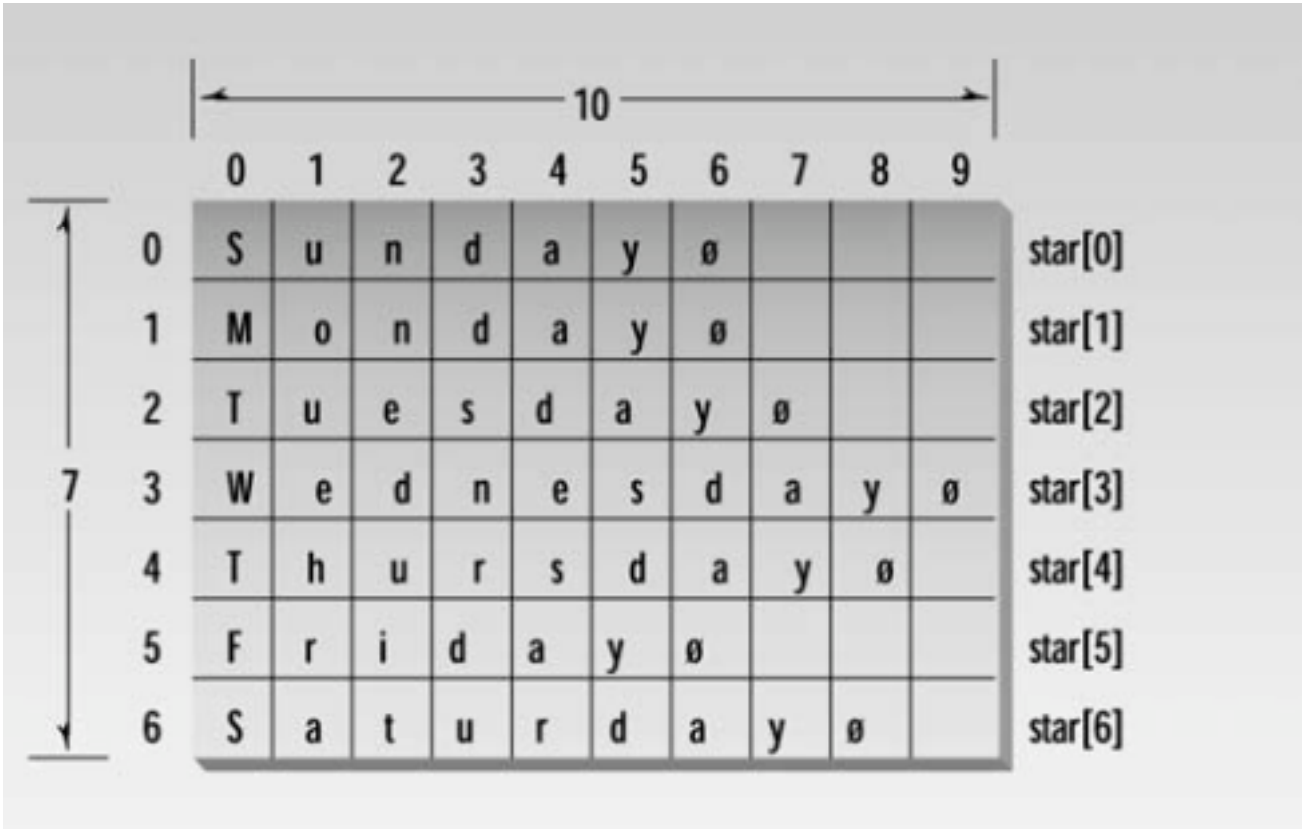
Array of Strings

```
int main()
{
    const int DAYS = 7;
    const int MAX = 10;
    char star[DAYS][MAX] = { “Sunday”, “Monday”,
          “Tuesday”, “Wednesday”, “Thursday”, “Friday”,
          “Saturday”  };
    for(int j=0; j<DAYS; j++)
          cout << star[j] << endl;
    return 0;
}
```

Array of Strings

- char star[DAYS][MAX] = { “Sunday”, “Monday”,
  “Tuesday”, “Wednesday”, “Thursday”,
  “Friday”, “Saturday” };
- The first dimension of this array, DAYS, tells how many strings are in the array.
- The second dimension, MAX, specifies maximum length of the strings (9 characters for “Wednesday” plus terminating null makes 10).
- Some bytes are wasted following strings that are less than the maximum length.

Array of Strings



String as Class Member

```
class part {
private:
      char partname[30];
      int partnumber;
      double cost;
public:
   void setpart(char pname[], int pn, double c)  {
               strcpy(partname, pname);
               partnumber = pn;
               cost = c;
}
   void showpart()   {
               cout << "\nName="   << partname;
               cout << ", number=" << partnumber;
               cout << ", cost=$"  << cost;
}
};
```

```
int main()    {
   part part1, part2;
   part1.setpart("handle bolt", 4473, 217.55);
   part2.setpart("start lever", 9924, 419.25);
   cout << "\nFirst part: ";
   part1.showpart();
   cout << "\nSecond part: ";
   part2.showpart();
   cout << endl;
   return 0;
}
```

User Defined String Type

- There are problems with C-String Type. For ex. You can not use the following expression to set one string equal to another
  - strDest = strSrc;
- The Standard C++ string class allows it.
- Goal: Creating our own string class which allow operation of the Standard C++ string class.

User Defined String Type

```
class String      {
    private:
        enum { SZ = 80; };
        char str[SZ];
    public:
        String()     { str[0] = '\0'; }
        String( char s[] )  { strcpy(str, s); }
        void display() { cout << str; }
        void concat(String s2)     {
                if( strlen(str)+strlen(s2.str) < SZ )
                        strcat(str, s2.str);
                else
                        cout << "\nString too long";
        }
};
```

User Defined String Type

```
int main()    {
    String s1("Merry Christmas!  ");
    String s2 = "Season's Greetings!";
    String s3;
    cout << "\ns1="; s1.display();
    cout << "\ns2="; s2.display();
    cout << "\ns3="; s3.display();
    s3 = s1;                        //assignment
    cout << "\ns3="; s3.display();
    s3.concat(s2);
    cout << "\ns3="; s3.display();
    cout << endl;
    return 0;
}
```

User Defined String Type

- The following syntax can be used to call one argument constructor
  - String s1(“Merry Christmas! “);
  - String s1 = “Merry Christmas! “;

Standard C++ String Class

- Standard C++ includes a new class called string.
- The string class assumes all the responsibility for memory management.
  - no longer need to worry about creating an array of the right size to hold string variables.
- The string class allows the use of overloaded operators, so you can concatenate string objects with the + operator:
  - s3 = s1 + s2

Defining and Assigning string Objects

```
#include <string>
int main()     {
      string s1("Man");              //initialize
      string s2 = "Beast";           //initialize
      string s3;                     //  creates an empty string variable
      s3 = s1;                      //assign
      cout << "s3 = " << s3 << endl;
      s3 = "Neither " + s1 + " nor ";  //concatenate
      s3 += s2;                      //concatenate
      cout << "s3 = " << s3 << endl;
      s1.swap(s2);                   //swap s1 and s2
      cout << s1 << " nor " << s2 << endl;
      return 0;
}
```

Defining and Assigning string Objects

- The overloaded + operator concatenates one string object with another.
  - s3 = “Neither “ + s1 + “ nor “;
    - places the string “Neither Man nor “ in the variable s3.
- Use the += operator to append a string to the end of an existing string
  - s3 += s2;
    - appends s2, which is “Beast”, to the end of s3, producing the string “Neither Man nor Beast” and assigning it to s3.
- string class member function: swap(), which exchanges the values of two string objects.

Input/Output with string Objects

- The << and >> operators are overloaded to handle string objects
- Function getline() handles input that contains embedded blanks or multiple lines.
  - This function is similar to the get() function used with C-strings, but is not a member function.
  - its first argument is the stream object from which the input will come (cin), and the second is the string object where the text will be placed, fullname.

```
int main()    {
      string fullname, nickname, address;
      string greeting("Hello, ");
      cout << "Enter your full name: ";
      getline(cin, fullname);       //reads embedded blanks
      cout << "Your full name is: " << fullname << endl;
      cout << "Enter your nickname: ";
      cin >> nickname;                    //input to string object
      greeting += nickname;         //append name to greeting
      cout << greeting << endl;
      cout << "Enter your address on separate lines\n";
      cout << "Terminate with '$'\n";
      getline(cin, address, '$');    //reads multiple lines
      cout << "Your address is: " << address << endl;
      return 0;                  }
```

Finding string Objects

- The string class includes a variety of member functions for finding strings and substrings in string objects.
- The find() function looks for the string used as its argument in the string for which it was called.
- The find_first_of() function looks for any of a group of characters, and returns the position of the first one it finds.
- The function find_first_not_of() finds the first character in its string that is not one of a specified group.

Finding string Objects

```
int main()    {
    string s1 = “In Xanadu did Kubla Kahn a stately pleasure dome
    decree”;
    int n;
    n = s1.find(“Kubla”);
    cout << “Found Kubla at “ << n << endl;
    n = s1.find_first_of(“spde”);
    cout << “First of spde at “ << n << endl;
    n = s1.find_first_not_of(“aeiouAEIOU”);
    cout << “First consonant at “ << n << endl;
    return 0;
}
```

Modifying string Objects

- The string class member functions erase(), replace(), and insert().
- The erase() function removes a substring from a string.
- The replace() function replaces part of the string with another string.
- The insert() function inserts the string specified by its second argument at the location specified by its first argument.

```
int main()    {
    string s1("Quick! Send for Count Graystone.");
    string s2("Lord");
    string s3("Don't ");
    s1.erase(0, 7);                 //remove chars from 0 to 7 - "Quick! "
    s1.replace(9, 5, s2);           //Go to position 9. replace 5 chars - "Count" by "Lord"
    s1.replace(0, 1, "s");          //Go to position 0 – "S", replace 'S' with 's'

    s1.insert(0, s3);               //Go to position 0 - insert "Don't " at beginning
    s1.erase(s1.size()-1, 1);     //Go to last position and remove '.'
    s1.append(3, '!');              //append "!" three times – "!!!"

    int x = s1.find(' ');           //find a space
    while( x < s1.size() )          //loop while spaces remain
    {
        s1.replace(x, 1, "/");     //replace with slash
        x = s1.find(' ');           //find next space
    }
    cout << "s1: " << s1 << endl;
    return 0;
}
```