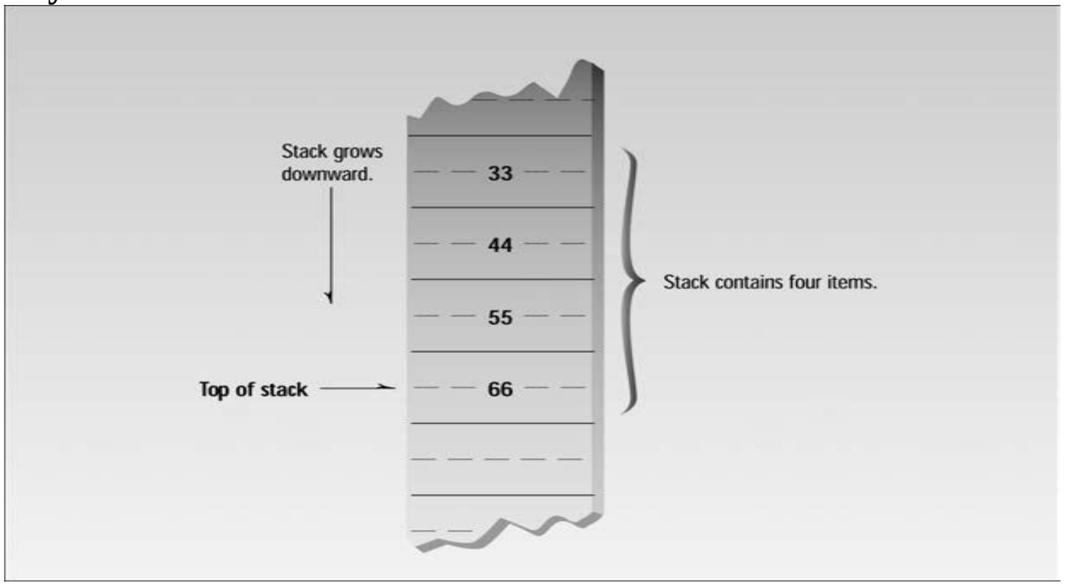
5.6 Arrays as Class Member Data

Arrays can be used as data items in classes. Let's look at an example that models a common computer data structure: the stack. A stack works like the spring-loaded devices that hold trays in cafeterias. When you put a tray on top, the stack sinks down a little; when you take a tray off, it pops up. The last tray placed on the stack is always the first tray removed.



A stack

```
#include <iostream.h>
class Stack
{
private:
enum { MAX = 10 }; //(non-standard syntax)
int st[MAX]; //stack: array of integers
int top; //number of top of stack
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;}
```

The size of the array used for the stack is specified by MAX, in the statement
enum { MAX = 10 };

In keeping with the philosophy of encapsulation, it’s preferable to define constants that will be used entirely within a class, as MAX is here, within the class. Thus the use of global const variables for this purpose is nonoptimal. Standard C++ mandates that we should be able to declare MAX within the class as static const int MAX = 10;
This means that MAX is constant and applies to all objects in the class. Unfortunately, some compilers, including the current version of Microsoft Visual C++, do not allow this newly approved construction.
Here’s the output:

```
1: 22
2: 11
3: 66
4: 55
5: 44
6: 33
```

5.6.1 Arrays of Objects

We’ve seen how an object can contain an array. We can also reverse that situation and create an array of objects. We’ll look at two situations: an array of English distances and a deck of cards.

5.6.2 Arrays of English Distances

The next program, ENGLARAY, demonstrates an array of such objects.

```
#include <iostream.h>
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() const //display distance
{ cout << feet << "\'-" << inches << '\"'; }
};
int main()
{Distance dist[100]; //array of distances
int n=0; //count the entries
char ans; //user response ('y' or 'n')
cout << endl;
do { //get distances from user
cout << "Enter distance number " << n+1;
dist[n++].getdist(); //store distance in array
cout << "Enter another (y/n)?: ";
cin >> ans;
} while( ans != 'n' ); //quit if user types 'n'
for(int j=0; j<n; j++) //display all distances
{cout << "\nDistance number " << j+1 << " is ";
dist[j].showdist();}
cout << endl;
return 0; }
```

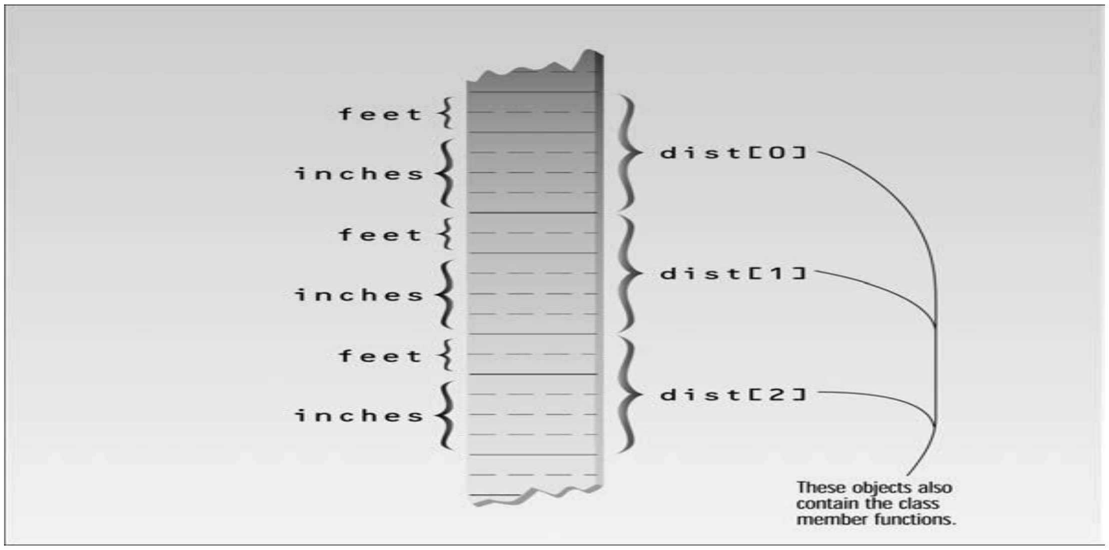
Here's a sample interaction when the user enters three distances:

```
Enter distance number 1
Enter feet: 5
Enter inches: 4
Enter another (y/n)? y
Enter distance number 2
Enter feet: 6
Enter inches: 2.5
Enter another (y/n)? y
Enter distance number 3
Enter feet: 5
Enter inches: 10.75
Enter another (y/n)? n
Distance number 1 is 5'-4"
Distance number 2 is 6'-2.5"
Distance number 3 is 5'-10.75"
```

5.6.3 Accessing Objects in an Array

Here's how the showdist() member function of the jth element of the array dist is invoked: dist[j].showdist();

As you can see, 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. This is similar to accessing a structure (or class) data member, except that the function name and parentheses are used instead of the data name. Notice that when we call the getdist() member function to put a distance into the array, we take the opportunity to increment the array index n: dist[n++]. getdist();



Array of objects.

5.7 String Variables

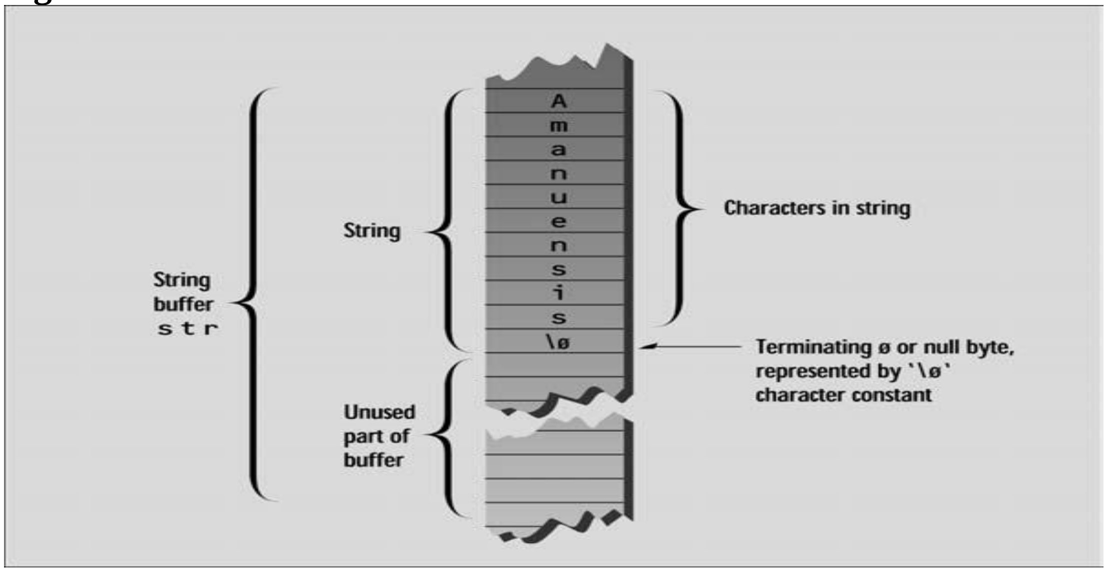
As with other data types, strings can be variables or constants. We'll look at these two entities before going on to examine more complex string operations. Here's an example that defines a single string variable. It asks the user to enter a string, and places this string in the string variable. Then it displays the string. Here's the listing for STRINGIN:

```
#include <iostream.h>
int main()
{const int MAX = 80; //max characters in string
char str[MAX]; //string variable str
cout << "Enter a string: ";
cin >> str; //put string in str
cout << "You entered: " << str << endl;
return 0;}
```

The definition of the string variable str looks like (and is) the definition of an array of type char:

char str[MAX];

We use the extraction operator >> to read a string from the keyboard and place it in the string variable str. This operator knows how to deal with strings; it understands that they are arrays of characters. If the user enters the string "Amanuensis" (one employed to copy manuscripts) in this program, the array str will look something like Figure bellow:



String stored in string variable.

5.7.1 String Constants

You can initialize a string to a constant value when you define it. Here's an example, STRINIT,

that does just that (with the first line of a Shakespearean sonnet):

```
#include <iostream.h>
int main()
{char str[ ] = "Farewell! thou art too dear for my possessing.";
cout << str << endl;
return 0;}
```

5.7.2 Reading Embedded Blanks

If you tried the STRINGIN program with strings that contained more than one word, you may have had an unpleasant surprise. Here's an example:

```
Enter a string: Law is a bottomless pit.
  You entered: Law
```

Wher\e did the rest of the phrase (a quotation from the Scottish writer John Arbuthnot, 1667– 1735) go? It turns out that the extraction operator >> considers a space to be a terminating character. Thus it will read strings consisting of a single word, but anything typed after a space is thrown away. To read text containing blanks we use another function, cin.get(). This syntax means a member function get() of the stream class of which cin is an object. The following example, BLANKSIN, shows how it's used.

```
#include <iostream.h>
int main()
{
const int MAX = 80; //max characters in string
char str[MAX]; //string variable str
cout << "\nEnter a string: ";
cin.get(str, MAX); //put string in str
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, thus automatically avoiding buffer overrun. Using this function, the input string is now stored in its entirety.

```
Enter a string: Law is a bottomless pit.
  You entered: Law is a bottomless pit.
```

5.8 Strings as Class Members

Strings frequently appear as members of classes. The next example, uses a string to hold the name of the widget part.

```
// strpart.cpp
// string used in widget part object
#include <iostream.h>
#include <cstring.h> //for strcpy()
class part
{private:
char partname[30]; //name of widget part
int partnumber; //ID number of widget part
double cost; //cost of part
public:
void setpart(char pname[], int pn, double c)
{strcpy(partname, pname);
partnumber = pn;
cost = c;}
void showpart() //display data
{cout << “\nName=” << partname;
cout << “, number=” << partnumber;
cout << “, cost=$” << cost;}
};
int main()
{
part part1, part2;
part1.setpart(“handle bolt”, 4473, 217.55); //set parts
part2.setpart(“start lever”, 9924, 419.25);
cout << “\nFirst part: “; part1.showpart(); //show parts
cout << “\nSecond part: “; part2.showpart();
cout << endl;
return 0;
}
```

This program defines two objects of class part and gives them values with the setpart() member function. Then it displays them with the showpart() member function. Here’s the output:

```
First part:
Name=handle bolt, number=4473, cost=$217.55
Second part:
Name=start lever, nu
```