Secure Coding

3 ARRAYS AND STRING

- **3.1 Arrays**
- **3.2 String**

3.1 Arrays
Arrays

- **Arrays**

Arrays are a kind of composite type because they allow you to clump a lot of variables together, one right after the other, under a single identifier name.

3.1 Arrays
Arrays

Syntax:

type ArrayName[constant expression];

Example:

int a[10];

You create storage for 10 int variables stacked on top of each other. They are all lumped under the name a.

3.1 Arrays
Arrays

To access one of these array elements, you use the same square-bracket syntax that you use to define an array:

```
a[5] = 47;
```

3.1 Arrays
Arrays

However, you must remember that even though the size of a is 10, you select array elements starting at zero ,so you can select only the array elements 0-9.

3.1 Arrays
Arrays

```
#include <iostream>
using namespace std;
int main( )
{
  int a[10];
  for(int i = 0; i < 10; i++)
  {
    a[i] = i * 10;
    cout << "a[" << i << "] = " << a[i] << endl;
  }
}
```

3.1 Arrays
Arrays

Remarks:

A) The memory address location of the elements in the array is consecutive.

B) The name of the array is the memory address location of the first element in the array.

C) The name of the array is a constant.

3.1 Arrays
Arrays

a:	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]

3.1 Arrays
Arrays

- **Initialization**

There are many ways of initializing the arrays.

1) static int a[10]={0,1,2,3,4,5,6,7,8,9};

2) static int a[10]={0,1,2,3,4};

3) static int a[ ]={1,2,3,4,5}

3.1 Arrays
Arrays

- **Two Dimensional Arrays**

Syntax:

type ArrayName[constant expression] [constant expression];

Example:

int a[10][5];

You create storage for 50, 10*5, int variables in an array.

3.1 Arrays
Arrays

Initialization:

1) static int a[3][4]={1,2,3,4,5,6,7,8,9,10,11,12};

2) static int
a[3][4]={{1,2,3,4},{5,6,7,8},{9,10,11,12}};

3) static int a[3][4]={{1},{0,6},{0,0,11}};

3.1 Arrays
Arrays

- **Array Argument**

The fact that naming an array produces its starting address turns out to be quite important when you want to pass an array to a function.

3.1 Arrays
Arrays

If you declare an array as a function argument, what you're really declaring is a **pointer**.

Arrays can't be passed by value, that is, you never automatically get a local copy of the array that you pass into a function.

When you modify an array, you're always **modifying** the outside object.

3.1 Arrays
Arrays

Example:

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

void func1(int a[ ], int size)
{
  for(int i = 0; i < size; i++)
    a[i] = i * i - i;
}
```

```
void print(int a[ ], string name, int size)
{
  for(int i = 0; i < size; i++)
    cout << name
         << "[" << i << "] = "
         << a[i] << endl;
}
```

3.1 Arrays
Arrays

```
int main()
{
  int a[5];
  // Probably garbage values:
  print(a, "a", 5);
  // Initialize the arrays:
  func1(a, 5);
  print(a, "a", 5);
  return 0;
}
```

Output:

```
a[0] = -858993460
a[1] = -858993460
a[2] = -858993460
a[3] = -858993460
a[4] = -858993460
a[0] = 0
a[1] = 1
a[2] = 2
a[3] = 3
a[4] = 4
```

3.1 Arrays
Object Arrays

- **Define Arrays**

Syntax:

ClassName ArrayName[constant expression];

Example:

```
Time t[10];
t[0].SetTime(1900,1,1);
```

You create storage for 10 Time variables in an array.

3.1 Arrays
Object Arrays

Initialization

1) Time t[2]={Time(1900,1,1), Time(1900,2,2)};

2) Time t[2];

3.1 Arrays
Object Arrays

Remarks:

A) Every element in the array calls the constructor automatically.

B) If there is not a user defined constructor, the C++ system will provide a default constructor.

C) The constructor can be default argument function, too.

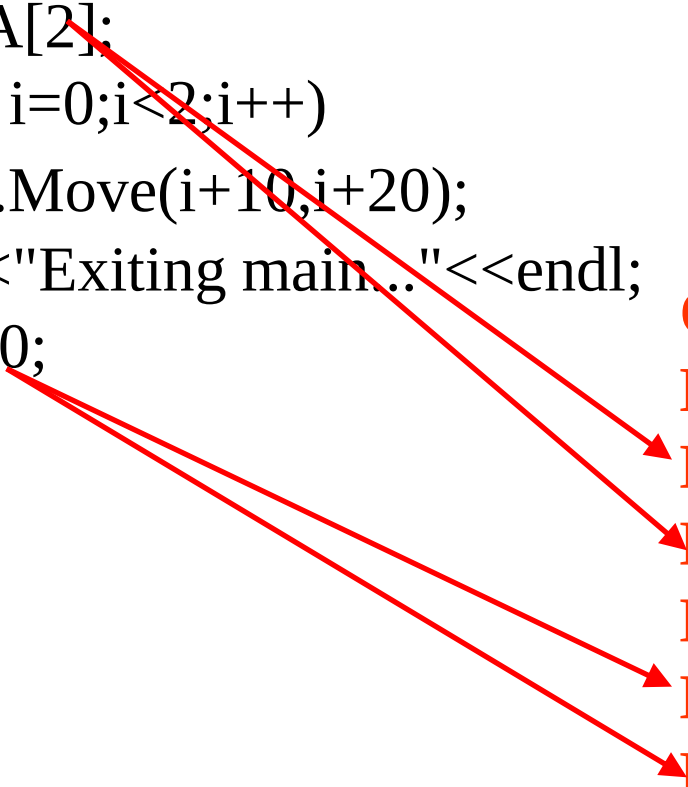
D) Every element calls the destructor when it is delete.

Example:

```
//Point.h
#if !defined(_POINT_H)
#define _POINT_H
class Point
{
public:
        Point( );
        Point(int xx,int yy);
        ~Point( );
        void Move(int x,int y);
        int GetX( ) {return X;}
        int GetY( ) {return Y;}
private:
        int  X,Y;
};
#endif
```

```
//Point.cpp
#include<iostream>
using namespace std;
#include "Point.h"
Point::Point( )
{
    X=Y=0;
    cout<<"Default Constructor called."<<endl;
}
Point::Point(int xx,int yy)
{
    X=xx;
    Y=yy;
    cout<< "Constructor called."<<endl;
}
Point ::~Point( )
{   cout<<"Destructor called."<<endl;  }
void Point ::Move(int x,int y)
{    X=x;    Y=y;  }
```

```
#include<iostream>
#include "Point.h"
using namespace std;
int main( )
{
    cout<<"Entering main..."<<endl;
    Point A[2];
    for(int i=0;i<2;i++)
       A[i].Move(i+10,i+20);
    cout<<"Exiting main..."<<endl;
    return 0;
}
```



Output:

Entering main...

Default Constructor called.

Default Constructor called.

Exiting main...

Destructor called.

Destructor called.

3.2 Strings
Arrays of Char

In C++ arrays of char are useful because there is not a built-in data type to store a string.

Remark:

A string stored by array of char ends by ‘\0’.

3.2 Strings
Arrays of Char

Example:

```
static char str[8]={112,114,111,103,114,97,109,0};
static char str[8]={'p','r','o','g','r','a','m','\0'};
static char str[8]="program";
static char str[]="program";
```

3.2 Strings
Arrays of Char

Example:

```
#include<iostream>
using namespace std;
void main( )
{
   static char c[10]={'I',' ','a','m',' ','a',' ','b','o','y'};
   int i;
   for(i=0;i<10;i++)
         cout<<c[i];
   cout<<endl;
}
```

Output:
I am a boy

3.2 Strings
Arrays of Char

- **Input and Output**

We can input or output a string by array of char at once.

Example:

```
char c[]="China";
char a[10];
cout<<c;
cin>>a;
```

3.2 Strings
Arrays of Char

Remarks:

A) The characters output by array don’t include ‘\0’.

B) Space is not included in an array of char.

3.2 Strings
Arrays of Char

Example:

```
static char str1[5],str2[5],str3[5];
cin>>str1>>str2>>str3;
```

Input:

How are you?

Memory:

str1: How\0

str2: are\0

str3: you?\0

3.2 Strings
Arrays of Char

Example:

```
static char str[13];
cin>>str;
```

Input:

How are you?

Memory:

str1: How\0

3.2 Strings
Arrays of Char

Example:

```
char a[4], *p1, *p2;
a="abc";        // compile error
cin>>p1;        //run time error

p1="abc";
p2=a;
cin>>p2;
```

3.2 Strings
Arrays of Char

- **getline and get**

cin.getline (char* pch, int nCount, char delim = '\n');

Parameters:

pch:

A pointer to a character array.

nCount:

The maximum number of characters to store, including the terminating NULL.

delim:

The delimiter character (defaults to newline).

3.2 Strings
Arrays of Char

Remarks:

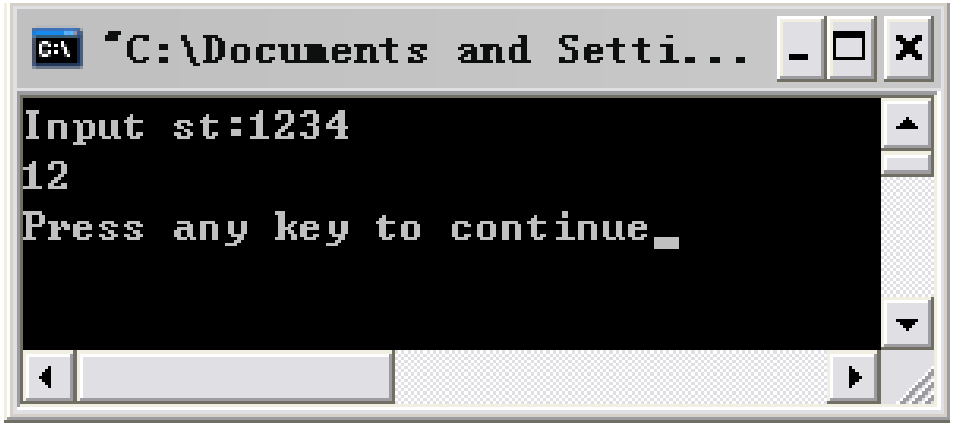
Extracts characters from the stream until either the delimiter delim is found, the limit nCount–1 is reached, or end of file is reached.

The characters are stored in the specified array followed by a null terminator.

If the delimiter is found, it is extracted but not stored.

3.2 Strings
Arrays of Char

```
#include <iostream>
using namespace std;
void main(void)
{
      char st[5] ;
      cout<<"Input st:";
      cin.getline(st,5,'3');
      cout<<st<<endl;
}
```



3.2 Strings
Arrays of Char

cin.get(char* pch, int nCount, char delim = '\n');

Parameters:

pch:

A pointer to a character array.

nCount:

The maximum number of characters to store, including the terminating NULL.

delim:

The delimiter character (defaults to newline).

3.2 Strings
Arrays of Char

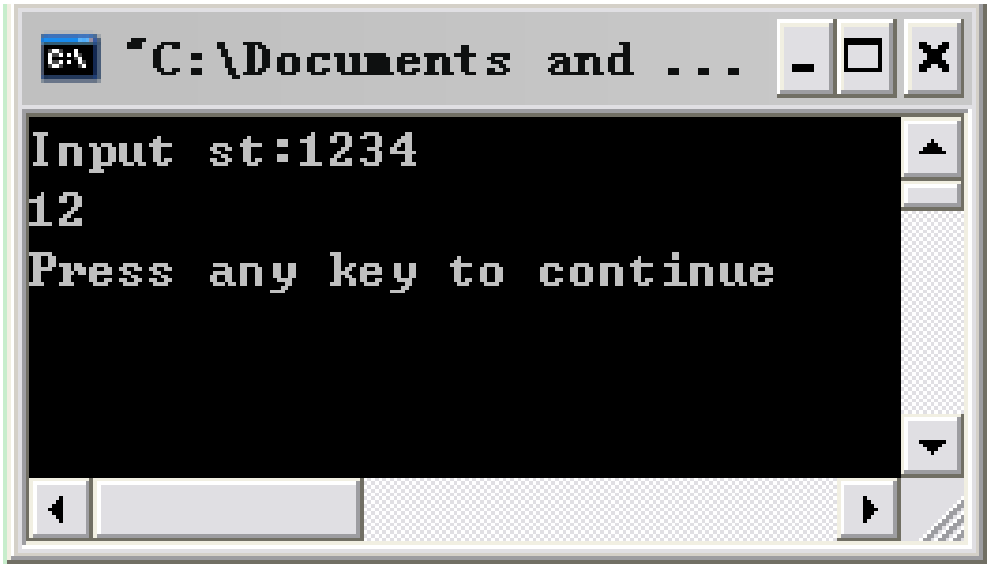
Remark:

Extracts characters from the stream until either delim is found, the limit nCount is reached, or the end of file is reached.

The characters are stored in the array followed by a null terminator.

3.2 Strings
Arrays of Char

```
#include <iostream>
using namespace std;
void main(void)
{
    char st[5] ;
    cout<<"Input st:";
    cin.get(st, 3, '3');
    cout<<st<<endl;
}
```



Difference between get and getline

get leaves the terminating character in the stream

getline removes the terminating character

3.2 Strings
Arrays of Char

Example:

```
#include <iostream>
using namespace std;
void main (void)
{
    char city[80];
    char state[80];
    int  i;
    for (i = 0; i < 2; i++)
    {
        cin.getline(city,80,',');
        cin.getline(state,80,'\n');
        cout << "City: " << city << "   State: "
        << state << endl;
    }
}
```

3.2 Strings
Arrays of Char

Input:

Beijing,China

Shanghai,China

Output:

City: Beijing Country: China

City: Shanghai Country: China

3.2 Strings
Arrays of Char

- **Functions**

```
char *strcat( char *strDestination, const char *strSource );
char *strcpy( char *strDestination, const char *strSource );
int    strcmp( const char *string1, const char *string2 );
char *strlen( const char *string );
char *_strlwr( char *string );
char *_strupr( char *string );
```

3.2 Strings
Strings

- **Definition**

C++ furnishes the type string as an alternative to C's null-terminated arrays of char.

Use of type string requires the header string.

3.2 Strings
Strings

Example:

```
#include <string>
using namespace std;
string s1;
string s2=”Bravo”;
string s3=s2;
string s4(10,’x’);
```

3.2 Strings
Strings

- **Input and Output**

Operator >>and <<can be used for input and output of strings.

3.2 Strings
Strings

Example:>>

```
string s;
cin>>s;
```

Input:

```
Ed Wood
```

Result:

```
string s: Ed
```

3.2 Strings
Strings

Example:<<

```
string s1;
string s2=”Bravo”;
string s3=s2;
string s4(10,’x’);
cout<<s1<<’\n’
      <<s2<<’\n’
      <<s3<<’\n’
      <<s4<<’\n’;
```

Output:

```
Bravo
Bravo
xxxxxxxxxx
```

3.2 Strings
Strings

- **Assignment and Concatenation**

The assignment operator = can be used to perform string assignments.

3.2 Strings
Strings

Example:

```
string s1,s2;
s1=”Ray Dennis Steckler”;
s2=s1;
cout<<s1<<endl;
cou<<s2<<endl;
```

Output:

```
Ray Dennis Steckler
Ray Dennis Steckler
```

3.2 Strings
Strings

Operators + and += can be used to perform string concatenation.

Example:

```
string s1=”Atlas”;
string s2=”King”;
string s3;
s3=s1+s2;
cout<<s3<<endl;
s1+=s2;
cout<<s1;
```

Output:

```
Atlas King
Atlas King
```

3.2 Strings
Strings

- **Extracting a Substring**

Example:

```
string s1=”Ray Dennis Stechler”;
string s2;
s2=s1.substr(4,6);
cout<<s2;
```

Output:

```
Dennis
```

3.2 Strings
Strings

Searching

Syntax:

s1.find(s2, ind);

If s2 is a substring of s1 at index ind or higher, find returns the smallest index>=ind where s2 begins.

3.2 Strings
Strings

Example:

```
string s1=”Ray Dennis Stechler”;
string s2=” Dennis”;
int f;
f=s1.find(s2);
if(f<s1.length())
        cout<<”Found at index:”<<f<<endl;
else
        cout<<”Not found”<<endl;
```

Output:

```
Found at index:4
```

3.2 Strings
Strings

- **Comparing Strings**

The operators ==, !=, <, <=, > and >= can be used for string comparison.

Example:

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

void trueFalse(int x)
{
    cout << (x? "True": "False") << endl;
}
```

```
void main()
{
        string S1="DEF",  S2="123";
        char CP1[ ]="ABC";
        char CP2[ ]="DEF";
        cout << "S1 is " << S1 << endl;
        cout << "S2 is " << S2 << endl;
        cout<<"length of S2:“
            <<S2.length()<<endl;
        cout << "CP1 is " << CP1 << endl;
        cout << "CP2 is " << CP2 << endl;
        cout << "S1<=CP1 returned ";
        trueFalse(S1<=CP1);
        cout << "CP2<=S1 returned ";
        trueFalse(CP2<=S1);
        S2+=S1;
        cout<<"S2=S2+S1:"<<S2<<endl;
        cout<<"length of S2:"<<S2.length()<<endl;
}
```

Output:

```
S1 is DEF
S2 is 123
length of S2:3
CP1 is ABC
CP2 is DEF
S1<=CP1 returned False
CP2<=S1 returned True
S2=S2+S1:123DEF
length of S2:6
```

Output:

S1 is DEF
S2 is 123
length of S2:3
CP1 is ABC
CP2 is DEF
S1<=CP1 returned False
CP2<=S1 returned True
S2=S2+S1:123DEF
length of S2:6

Summarize

- Arrays
- String