EE 441 Data Structures
Lecture 3
Arrays

Ilkay Ulusoy

Arrays

- An array is a consecutive group of memory locations (i.e., elements of the array).
- The contents of each element are of the same type.
  - Could be an array of int, double, char, …
- We can refer to individual elements by giving the position number (index, subscript) of the element in the array.

Declaring An Array

```
element_type array_name[number_of_elements];
```

`element_type` can be any C++ variable type.

`array_name` can be any valid variable name.

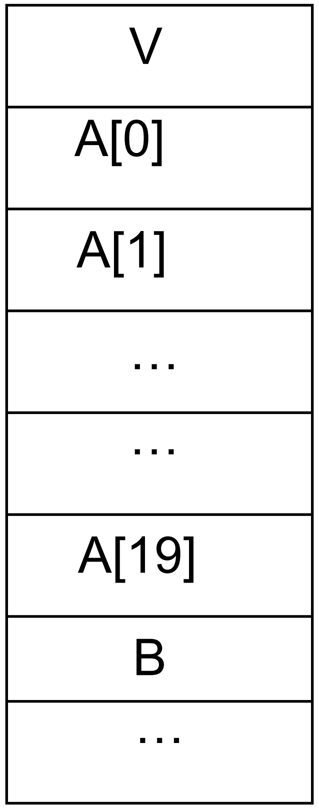
`number_of_elements` can be an expression.

Arrays

- Arrays in C++ are indexed from zero and *not one!!*
  - *The first element is the 0th element !*
  - If you declare an array of *n* elements, the last one is number *n-1*.
  - *If you try to access element number n it is an error !*
- Indexes are always integer, e.g., A[10]
- The number of elements in the array must be known at compile time, e.g., int j[3]

C++ Operations on Arrays

Most C++ compilers don’t check array index range !!



```
int V=20;
int A[20]; /*  index range is 0-19  */
int B;
```

- A[V]=0; /* index is out of range, but most C++ compilers don’t check this */
- Effect is B=0, since the first location after A is reserved for B by above decleration

Arrays

- Decleration

```
const int ArraySize=50;
/*int ArraySize=50; WILL NOT WORK, it has to be
  constant decleration*/
float      A[ArraySize];
long       X[Arraysize+10];
```

- Assignment

```
A[i]=z;
t=X[i];
X[i]=X[i+1]=t;
/* this is equivalent to X[i+1]=t;  X[i]=X[i+1] ,
  i.e. right to left */
```

Initialization

- You can initialize an array when you declare it (just like with variables):

```
int grades[5] = { 1,8,3,6,12};
double d[2] = { 0.707, 0.707};
char s[] = { ‘M’, ‘E’, ‘T’, `U’ };
```

You don’t need to specify a size when initializing, the compiler will count for you.

Array Example

```
int main(void)
{
    int facs[10];

    for (int i=0;i<10;i++)
          facs[i] = factorial(i);

    for (int i=0;i<10;i++)
          cout << "factorial(" << i << ") is " << facs[i] << endl;
}
```

Two-dimensional Arrays

```
int T[3][4]={{20,5,-3,0},{-85,35,-
  1,2},{15,3,-2,-6}};
int a=T[2][3];
T[0][4]=a;
```

e.g.,

T[2] [3]=-6

T[2][0]=15

T[0]	20	5	3	0
T[1]	-85	35	-1	2
T[2]	15	3	-2	-6

Class Declaration (Recap.)

```
Class   <ClassName>
{
    private:
    <private data declarations>
    <private method declarations>
    public:
            <public data declarations>
            <public method declarations>
}
```

Class Declaration (Recap.)

```
Class Rectangle
{
private:
    float length, width;
public:
    Rectangle (float l=0, float w=0); // constructor
    float GetLength(void) const;
    void PutLength (float l);
    float GetWidth(void) const;
    void PutWidth(float w);
    float Perimeter(void) const;
    float Area(void) const;
};
```

Arrays of Objects

```
Rectangle room[100];
```

// constructor is called for room[0] ..room[99];

- This declaration creates an array of 100 objects. Each have different members, all (in this case) initialized to the default values.
- For declaring large array objects, a constructor with default values or with no parameters is preferred!
- Rectangle room[3]={Rectangle(10,15), Rectangle(5,8), Rectangle(2,30)}; may be practical, but
- Rectangle room[100]; simply initializes all length and with values to the default value of 0.

Arrays of characters and Strings

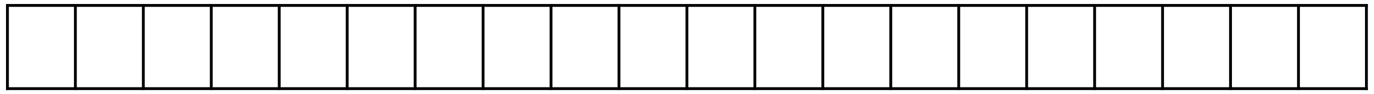
C++ Standard Library implements a **string** class, which is useful to handle and manipulate strings of characters.

Since strings are in fact sequences of characters, we can represent them also as plain arrays of char elements.

For example, declare Jenny as an array that can store up to 20 elements of type char:

char Jenny [20];

It can be represented as:

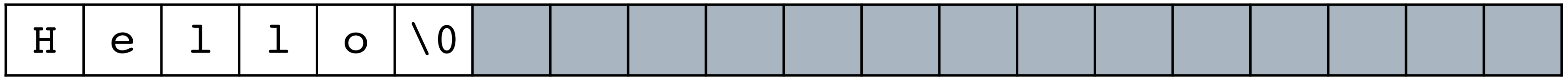
Jenny:


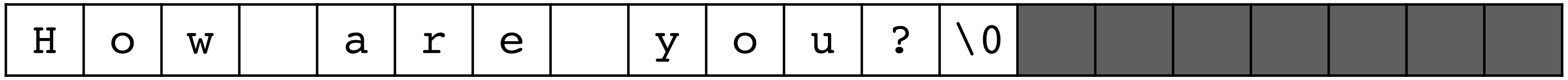
In the array Jenny, in theory, we can store sequences of characters up to 20 characters long. But we can also store shorter sequences.

A special character is used to signal the end of the valid sequence: the *null character*, whose literal constant can be written as '\0' (backslash, zero).

Our array of 20 elements of type char, called jenny, can be represented storing the characters sequences "Hello" and ”How are you?" as:

Jennny:





The panels in gray color represent char elements with undetermined values.

Initialization of null-terminated character sequences

char myword[] = { 'H', 'e', 'l', 'l', 'o', '\0' };

In this case we would have declared an array of 6 elements of type char initialized with the characters that form the word "Hello" plus a null character '\0' at the end.

But arrays of char elements have an additional method to initialize their values: using string literals.

char myword [ ] = "Hello";

So string literals enclosed between double quotes always have a null character ('\0') automatically appended at the end.

In both cases the array of characters myword is declared with a size of 6 elements of type char: the 5 characters that compose the word "Hello" plus a final null character ('\0') which specifies the end of the sequence.

Notice that we are talking about initializing an array of characters in the moment it is being declared, and not about assigning values to them once they have already been declared. In fact because this type of null-terminated arrays of characters are regular arrays we have the same restrictions that we have with any other array, so we are not able to copy blocks of data with an assignment operation.

Assuming mystext is a char[ ] variable declared previously. Assignment operations within a source code like:

```
mystext = "Hello";
 mystext[ ] = "Hello";
mystext = { 'H', 'e', 'l', 'l', 'o', '\0' };
```

would **not** be valid.

- cin and cout support null-terminated sequences as valid containers for sequences of characters, so they can be used directly to extract strings of characters from cin or to insert them into cout.

For example:

```
// null-terminated sequences of characters
#include <iostream>;
voidmain ()
{ char question[] = "Please, enter your first name: ";
char greeting[] = "Hello, ”;
char yourname [80];
cout << question; cin >> yourname;
cout << greeting << yourname << "!”;}
```

```
a sample run:
Please, enter your first name: John
Hello, John!
```

Three arrays of char elements are declared:

- The first two were initialized with string literal constants
- The third one was left uninitialized.

In any case, we have to specify the size of the array:

- in the first two (question and greeting) the size was implicitly defined by the length of the literal constant they were initialized to.
- While for yourname we have explicitly specified that it has a size of 80 chars.

Finally, sequences of characters stored in char arrays can easily be converted into string objects just by using the assignment operator:

```
string mystring;
char myntcs[ ]="some text";
mystring = myntcs;
```