Content

- One dimensional array
- char array and string
- String functions
- Two and more dimensional arrays
- Array of strings
- More examples for strings
One-Dimensional Arrays

- A one-dimensional array is a list of related variables. The general form of a one dimensional array declaration is
- \textit{type name}[size];
- Here, \textit{type} declares the base type of the array. The base type determines the data type of each element that comprises the array. \textit{size} defines how many elements the array will hold. For example, the following declares an integer array named \textit{sample} that is ten elements long:
- \textit{int sample}[10];
For example

```c
{   int i[7];
    int j;
    for(j=0; j<7; j++) i[j] = j; }
```

- Here, `i` looks like this:

```
   0     1     2     3     4     5     6
```

- For a one-dimensional array, the total size of an array in bytes is computed as shown here:

  - `total bytes = number of bytes in type × number of elements`
In C++, you cannot assign one array to another. For example, the following is illegal:
{
int a[10], b[10];
// ...

a = b; // error – illegal
}

To transfer the contents of one array into another, you must assign each value individually.
// An incorrect program. Do Not Execute!

int main()
{
    int crash[10], i;
    for(i=0; i<100; i++) crash[i] = i; // Error!
    return 1;
}

- In this case, the loop will iterate 100 times, even though crash is only ten elements long!
Strings

- In C++, a string is defined as a character array that is terminated by a null. A null character is specified using '\0', and is zero. Because of the null terminator, it is necessary to declare a character array to be one character longer than the largest string that it will hold.
- For example, if you want to declare an array `str` that could hold a 10-character string, you would write:
  
```
char str[11];
```
• A string literal is a list of characters enclosed in double quotes. Here are some examples:

1) "hello there" 2) "I like C++"

• The last string shown is " ". This is called a null string. It is not necessary to manually add the null onto the end of string constants; the C++ compiler does this for you automatically.

• Therefore, the string "Hello" will appear in memory like this:
Reading a String from the Keyboard

```cpp
int main()
{
    char str[80];
    cout << "Enter a string: ";
    cin >> str; // read string from keyboard
    cout << "Here is your string: ";
    cout << str;
    return 0;
}
```

- Enter a string: This is a test
- Here is your string: This
/ Using gets() to read a string from the keyboard.
#include <cstdio>

int main()
{
    char str[80];
    cout << "Enter a string: ";
    gets(str); // read a string from the keyboard
    cout << "Here is your string: ";
    cout << str;
    return 0;
}
Some String Library Functions

- C++ supports a wide range of string-manipulation functions. The most common are:
  - `strcpy` 
  - `strcat` 
  - `strlen` 
  - `strcmp` 
  - `strncpy` 
  - The string functions all use the same header, `<cstring>`.

A call to `strcpy()` takes this general form:

- `strcpy(to, from);`

The `strcpy()` function copies the contents of the string `from` into `to`. Remember, the array that forms `to` must be large enough to hold the string contained in `from`. If it isn’t, the `to` array will be overrun, which will probably crash your program.
The following program will copy "hello" into string str:

```cpp
#include <cstring>

int main()
{
    char str[80];
    strcpy(str, "hello");
    cout << str;
    return 0;
}
```
strcat

- A call to `strcat()` takes this form:
- `strcat(s1, s2);`
- The `strcat()` function appends `s2` to the end of `s1`; `s2` is unchanged. Both strings must be null-terminated, and the result is null-terminated.
For example, the following program will print **hello there** on the screen:

```c
#include <cstring>

main()
{
    char s1[20], s2[10];
    strcpy(s1, "hello");
    strcpy(s2, "there");
    strcat(s1, s2);
    cout << s1;
}
```
**strcmp**

- A call to `strcmp()` takes this general form:
  
  ```c
  strcmp(s1, s2);
  ```

- The `strcmp()` function compares two strings and returns 0 if they are equal. If `s1` is greater than `s2` then a positive number is returned; if it is less than `s2`, a negative number is returned.
The **password()** function, shown in the following program, is a password-verification routine. It uses `strcmp()` to check a user’s input against a password.

```cpp
bool password();

int main()
{
    if(password())
        cout << "Logged on. \n";
    else
        cout << "Access denied. \n";
    return 0;
}

// Return true if password accepted; false otherwise.
bool password()
{
    char s[80];
    cout << "Enter password: ";
    gets(s);
    if(strcmp(s, "password"))
        cout << "Invalid password. \n";
    return false;
}
```

The key to using `strcmp()` is to remember that it returns false when the strings match.

```c
int main()
{
    char s[80];
    for(;;) {
        cout << "Enter a string: ";
        gets(s);
        if(!strcmp("quit", s)) break;
    }
    return 0;
}
```
**strlen**

- The general form of a call to `strlen()` is `strlen(s);`
- where `s` is a string. The `strlen()` function returns the length of the string pointed to by `s`. 
The following program will print the length of a string entered from the keyboard:

```c
int main()
{
    char str[80];
    cout << "Enter a string: ";
    gets(str);
    cout << "Length is: " << strlen(str);
    return 0;
}
```

If the user enters the string "Hi there", this program will display 8. The null terminator is not counted by `strlen()`.
Using the Null Terminator

// Convert a string to uppercase.
#include <cstring>
#include <cctype>

main()
{
    char str[80];
    int i;
    strcpy(str, "this is a test");
    for(i=0; str[i]!="\0"; i++) str[i] = toupper(str[i]); // You can use either '\0' or NULL,
    cout << str;
}
In addition to `toupper()`, the C++ standard library contains several other character-manipulation functions. For example, the complement to `toupper()` is `tolower()`, which returns the lowercase equivalent of its character argument. Other character functions include `isalpha()`, `isdigit()`, `isspace()`, and `ispunct()`. These functions each take a character argument and determine if it belongs to that category. For example, `isalpha()` returns true if its argument is a letter of the alphabet.
Two-Dimensional Arrays

- C++ allows multidimensional arrays. The simplest form of the multidimensional array is the two-dimensional array. A two-dimensional array is a list of one-dimensional arrays.

- `int twod[10][20];`
int main()
{
    int t, i, num[3][4];
    for(t=0; t<3; ++t) {
        for(i=0; i<4; ++i) {
            num[t][i] = (t*4)+i+1;
            cout << num[t][i] << ' ';
        }
        cout << '\n';
    }
    return 0;
}
In this example, `num[0][0]` will have the value 1, `num[0][1]` the value 2, `num[0][2]` the value 3, and so on. The value of `num[2][3]` will be 12. Conceptually, the array will look like that shown in the Figure below:
In the case of a two-dimensional array, you can use this formula to determine the number of bytes of memory that will be allocated:

\[
\text{bytes} = \text{row} \times \text{column} \times \text{number of bytes in type}
\]

Therefore, assuming two-byte integers, an integer array with dimensions 10,5 would have \(10 \times 5 \times 2 = 100\) bytes allocated.
Multidimensional Arrays

- C++ allows arrays with more than two dimensions. Here is the general form of a multidimensional array declaration:

  ```
  type name[size1][size2]...[sizeN];
  ```

- For example, the following declaration creates a $4 \times 10 \times 3$ integer array:

  ```
  int multidim[4][10][3]; // 4x10x3x2 = 240 bytes
  ```
Array Initialization

- C++ allows the initialization of arrays. The general form of array initialization is similar to that of other variables, as shown here:
  ```
  int i[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
  ```

- This means that `i[0]` will have the value 1, and `i[9]` will have the value 10.

- Character arrays that will hold strings allow a shorthand initialization that takes this form:
  ```
  char name[size] = "string";
  ```

- For example, the following code fragment initializes `str` to the phrase "hello":
  ```
  char str[6] = "hello";
  ```

- This is the same as writing
  ```
  char str[6] = { 'h', 'e', 'l', 'l', 'o', '\0' };
  ```

- Because strings in C++ must end with a null, you must make sure that the array you declare is long enough to include it.
Multidimensional arrays are initialized in the same way as one-dimensional arrays.

For example, the following program initializes an array called `sqrsls` with the numbers 1 through 10 and their squares:

```c
int sqrs[10][2] = {
    1, 1,
    2, 4,
    3, 9,
    4, 16,
    5, 25,
    6, 36,
    7, 49,
    8, 64,
    9, 81,
    10, 100
};
```
When initializing a multidimensional array, you may add braces around the initializers for each dimension. This is called *subaggregate grouping*. For example, here is another way to write the preceding declaration:

```c
int sqrs[10][2] = {
  {1, 1},
  {2, 4},
  {3, 9},
  {4, 16},
  {5, 25},
  {6, 36},
  {7, 49},
  {8, 64},
  {9, 81},
  {10, 100}
};
```
Unsized Array Initializations

- `char e1[] = "Divide by 0\n";`
- `char e2[] = "End-of-File\n";`
- `char e3[] = "Access Denied\n";`

- Unsized array initializations are not restricted to one-dimensional arrays. For a multidimensional array, you must specify all but the leftmost dimension so that C++ can index the array properly. Using unsized array initializations, you can build tables of varying lengths, with the compiler automatically allocating enough storage for them.
• **Note:** It is important to remember that while initializing a 2-D array it is necessary to mention the second (column) dimension, whereas the first dimension (row) is optional.

• Thus the declarations,

```c
int arr[2][3] = { 12, 34, 23, 45, 56, 45 } ;
int arr[ ][3] = { 12, 34, 23, 45, 56, 45 } ;
```

are perfectly acceptable,

• whereas,

```c
int arr[2][ ] = { 12, 34, 23, 45, 56, 45 } ;
int arr[ ][ ] = { 12, 34, 23, 45, 56, 45 } ;
```

would never work.
Arrays of Strings

• For example, the following declares an array of 30 strings, each having a maximum length of 80 characters:

• char array[30][80];
String functions

- `strncpy`: `strncpy(str1,str2,n);` // `n` is number of alphabets you wish to copy
- `strstr`: `cout<<strstr(str1,str2);` // searches `str2` in the `str1`
- `atoi`: `num=stoi("4569");` // converts string into integer
- `atol`: `lnum=atol("500000");` // converts string into large integer
- `atof`: `fnum=atof("3.14");` // converts string into a float number
- `itoa`: `itoa(value, str1, base);` // converts integer into string
  // `base`(8=octal, 10=decimal, 16=hexadecimal, etc.)
More examples for strings

```cpp
char s1[] = "example";
char s2[20] = "another example"

• would store the two strings as follows:

s1 = | e | x | a | m | p | l | e | \0 |
s2 = | a | n | o | t | h | e | r | | e | x | a | m | p | l | e | \0 | ? | ? | ? | ? | ? |
```
• For example to read in a name in the form of a name followed by a surname we might use code as follows:

```c++
char name[12], surname[12];
cout << "Enter name ";
cin >> name;
cin >> surname;
cout << "The name entered was "
<< name << " " << surname;
```
C-strings vs C++ strings

A **C-string** is a zero terminated array of chars but the **C++ string** does without the extra terminating zero. Most likely it keeps track of the length internally but you should not make any assumptions about how the implementation works.

C-strings:  `#include <cstring>`
C++ strings:  `#include <string>`

Declaring a C-string variable:  `char str[10];`
Declaring a C++ string object:  `string str;`

Initializing a C-string variables:

- `char str2[] = "Send money!";`
- `char str3[] = {'O', 'K', '\0'};`
- `char str3[] = "OK";`

Initializing a C++ string objects:

- `string str1("Call home!");`
- `string str2 = "Send money!";`
- `string str3("OK");`
- `string str4(10, 'x');`
• Assigning to a C-string variable:
  char str[10];
  str = "Hello!";

• Assigning to a C++ string object:
  string str;
  str = "Hello!";
  Assigning other string:
  str = otherString;

• Concatenating two C-strings:
  strcat(str1, str2);
  strcpy(str, strcat(str1, str2));

• Concatenating two C++ string objects:
  str1 += str2;
  str = str1 + str2;
• Copying a C-string variable:
  char str[20];
  strcpy(str, "Hello!");
  strcpy(str, otherString);

• Copying a C++ string object:
  string str;
  str = "Hello";
  str = otherString;

• Accessing a single character of C-string variable:
  str[index];

• Accessing a single character of C++ string object:
  str[index];
  str.at(index);
  str(index, count);
• **Comparing two C-strings:**
  if (strcmp(str1, str2) < 0)
  cout << "str1 comes 1st.";
  if (strcmp(str1, str2) == 0)
  cout << "Equal strings.";
  if (strcmp(str1, str2) > 0)
  cout << "str2 comes 1st.";

• **Comparing two C++ string objects:**
  if (str1 < str2) cout << "str1 comes 1st.";
  if (str1 == str2) cout << "Equal strings.";
  if (str1 > str2) cout << "str2 comes 1st.";

• **Finding the length of a C-string:**
  strlen(str);

• **Finding the length of a C++ string:**
  str.length();
**Output of a C++ string object:**

```
cout << str;
cout << setw(width) << str; // cursor starts after width number of spaces
```

**Input of a C++ string object:**

```
cin >> s;
getline(cin, s);       // takes all like previous.
getline(cin, s, 'x');  // takes till x
```
• The `setw(width)` I/O manipulator can be used before outputting a string, the string will then be output right-justified in the field width. If the field width is less than the length of the string then the field width will be expanded to fit the string exactly.

• **Notes:**

  Keep in mind that `cin` ignores white space when reading a string, while `cin.get()`, `cin.getline()` and `getline()` do not.
1. 

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

string name;

cout << "Enter your name: " << flush;

// Or, alternatively:
getline(cin, name);  // read a whole line into the string name

if (name == "")
{
    cout << "You entered an empty string, " "assigning default \n";
    name = "John";
}
else
{
    cout << "Thank you, " "for running this simple program!" << endl;
}
```
2.

```c
string result;
string s1 = "hello ";
string s2 = "world";
result = s1 + s2;  // result contains "helloworld"
```

3.

```c
string result;
string s1 = "hello";  // without the extra space at the end
string s2 = "world";
result = s1;
result += ' ';   // append a space at the end
result += s2;   // result now contains "hello   world"
```
4.

string firstname, lastname, fullname;
cout << "First name: ";
getline (cin, firstname);
cout << "Last name: ";
getline (cin, lastname);
fullname = lastname + ", " + firstname;
cout << "Fullname: " << fullname << endl;
• **Reverse:**

```cpp
int main ()
{
    string str = "SDU";
    cout << str.reverse() << endl;
    return 0;
}
```

• **OUTPUT:**

UDS
• **Swap:**

```cpp
string str1 = "Robert";
string str2 = "Forest";
cout << "str1 is: " << str1 << endl;
cout << "str2 is: " << str2 << endl;
cout << "swap str1 with str2" << endl;
    str1.swap(str2);
cout << "str1 is: " << str1 << endl;
cout << "str2 is: " << str2 << endl;
```

• **OUTPUT:**

```
str1 is: Robert
str2 is: Forest
swap str1 with str2
str1 is: Forest
str2 is: Robert
```
```cpp
• **Length:**

```cpp
int main ()
{
    string str = "C++ is best computer language";
    cout << "str is: " << str << endl;
    cout << "Length of str is : " << str.length() << endl;
    return 0;
}
```

**OUTPUT:**

str is: C++ is best computer language
Length of str is : 29
Size:

```cpp
#include<string>

main()
{
    string str("We go to target");
    cout<<str<<endl;
    cout<<"size of str = "<<str.size()<<endl;
    system("pause");
}

OUTPUT:

str is: We go to target
size of str = 15
### C-String Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>strcpy(str1, str2)</code></td>
<td>Copies <code>str2</code> to <code>str1</code>.</td>
</tr>
<tr>
<td><code>strlen(str)</code></td>
<td>Returns the number of characters in <code>str</code>.</td>
</tr>
<tr>
<td><code>strcat(str1, str2)</code></td>
<td>Concatonates (joins) <code>str2</code> to <code>str1</code>.</td>
</tr>
<tr>
<td><code>strcmp(str1, str2)</code></td>
<td>Compares <code>str1</code> to <code>str2</code>.  &lt;br&gt;  If <code>str1 == str2</code> return 0. &lt;br&gt;  If <code>str1 &lt; str2</code> return -1. e.g. <code>strcmp(&quot;a&quot;, &quot;b&quot;)</code> &lt;br&gt;  If <code>str1 &gt; str2</code> return +1. e.g. <code>strcmp(&quot;b&quot;, &quot;a&quot;)</code></td>
</tr>
</tbody>
</table>

### C++ String Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>str.length()</code></td>
<td>Returns the number of characters in <code>str</code>.</td>
</tr>
<tr>
<td><code>str1.insert(pos, str2)</code></td>
<td>Insert <code>str2</code> into <code>str1</code> starting at <code>pos</code>.</td>
</tr>
<tr>
<td><code>str1.erase(pos, n)</code></td>
<td>Delete <code>n</code> characters in <code>str1</code> starting at <code>pos</code>.</td>
</tr>
<tr>
<td><code>str1.replace(pos, n, char)</code></td>
<td>Replace <code>n</code> characters in <code>str1</code> with <code>char x</code> starting at <code>pos</code>.</td>
</tr>
<tr>
<td><code>str1.append(str2)</code></td>
<td>Appends <code>str2</code> to <code>str1</code>.</td>
</tr>
<tr>
<td><code>str1.compare(str2)</code></td>
<td>Compares <code>str1</code> to <code>str2</code>. &lt;br&gt;  If <code>str1 == str2</code> return 0. &lt;br&gt;  If <code>str1 &lt; str2</code> return -1. e.g. <code>strcmp(&quot;a&quot;, &quot;b&quot;)</code> &lt;br&gt;  If <code>str1 &gt; str2</code> return +1. e.g. <code>strcmp(&quot;b&quot;, &quot;a&quot;)</code></td>
</tr>
<tr>
<td><code>str1.substr(pos, n)</code></td>
<td>Return <code>n</code> characters in <code>str1</code> starting at <code>pos</code>.</td>
</tr>
<tr>
<td><code>str1.swap(str2)</code></td>
<td>Exchange contents of <code>str1</code> and <code>str2</code>. &lt;br&gt;  (A useful function in programs that sort strings.)</td>
</tr>
</tbody>
</table>
• These are a lot of other functions to use for strings, but we covered which are used mostly.

THE END!