Programming Fundamentals

Aamina Batool

Character Array

- **Character array:** An array whose components are of type `char`

Null Character

- in C++, the null character is represented as **'\0'**,
- The statement:
- **char ch = '\0';**
- stores the null character in **ch**, where **ch** is a **char** variable
- the null character is less than any other character in the **char** data set.

C-string

- C-string is inherited from C language
- String is born in C++
- a string is a sequence of zero or more characters, and strings are enclosed in double quotation marks.
- In C++, **C**-strings are null terminated; that is, the last character in a **C**-string is always the null character.
- The most commonly used term for character arrays is **C**-strings
- **C**-strings are stored in (one-dimensional) character arrays

Difference b/w‘A’ & “A”

- The first one is character **`A`**; the second is **C**-string **`A`**.
- Because **C**-strings are null terminated, **`"A"`** represents two characters: **`'A'`** and **`'\0'`**.

How to store the string “A” in a character array?

- To store **'A'**, we need only one memory cell of type **char**;
- To store **"A"**, we need two memory cells of type **char**—one for **'A'** and one for **'\0'**

“Hello”

- the **C**-string
- **`"Hello"`** represents six characters: **`'H'`**, **`'e'`**, **`'l'`**, **`'l'`**, **`'o'`**, and **`'\0'`**
- to store the **C**-string **`"Hello"`** in computer memory, we need six memory cells of type **`char`**.

More Examples

- `char name[16];`
- This statement declares an array name of **16** components of type `char`. Because **C**-strings are null terminated and name has **16** components, the largest string that can be stored in **name** is of length **15**, to leave room for the terminating `'\0'`.

More Examples

- `char name[16] = {'J', 'o', 'h', 'n', '\0'};`
- declares an array **name** containing **16** components of type **char** and stores the **C**-string **"John"** in it.
- `char name[16] = "John";`
- The statement:
- `char name[] = "John";`
- declares a **C**-string variable name of a length large enough—in this case, **5**—and stores **"John"** in it.

Illegal operations

- `char studentName[26];`
- `studentName = "Lisa L. Johnson"; //illegal`

Built-in functions of C-String

Function	Effect
`strcpy(s1, s2)`	Copies the string `s2` into the string variable `s1` The length of `s1` should be at least as large as `s2` Does not check to make sure that `s1` is as large `s2`
`strncpy(s1, s2, limit)`	Copies the string `s2` into the string variable `s1`. At most `limit` characters are copied into `s1`.
`strcmp(s1, s2)`	Returns a value < 0 if `s1` is less than `s2` Returns 0 if `s1` and `s2` are the same Returns a value > 0 if `s1` is greater than `s2`
`strncmp(s1, s2, limit)`	This is same as the previou functions `strcmp`, except that at most `limit` characters are compared.
`strlen(s)`	Returns the length of the string `s`, excluding the null character

String Comparison

- In C++, **C**-strings are compared character by character using the system's collating sequence.
- The C-string **`"Air"`** is less than the C-string **`"Boat"`** because the first character of **`"Air"`** is less than the first character of **`"Boat"`**.
- The C-string **`"Air"`** is less than the C-string **`"An"`** because the first characters of both strings are the same, but the second character **`'i'`** of **`"Air"`** is less than the second character **`'n'`** of **`"An"`**.

String Comparison

- The C-string **"Bill"** is less than the C-string **"Billy"** because the first four characters of **"Bill"** and **"Billy"** are the same, but the fifth character of **"Bill"**, which is **'\0'** (the null character), is less than the fifth character of **"Billy"**, which is **'y'**. (Recall that **C**-strings in C11 are null terminated.)

- The C-string **"Hello"** is less than **"hello"** because the first character **'H'** of the **C**-string **"Hello"** is less than the first character **'h'** of the **C**-string **"hello"**.

- the function **strcmp** compares its first **C**-string argument with its second **C**-string argument character by character.

Use of built-in functions

- `char studentName[21];`
- `char myname[16];`
- `char yourname[16];`
- The following statements show how string functions work:

Statement	Effect
`strcpy(myname, "John Robinson");`	`myname = "John Robinson"`
`strlen("John Robinson");`	Returns 13, the length of the string `"John Robinson"`
`int len;` `len = strlen("Sunny Day");`	Stores 9 into `len`

Use of built-in functions

```
strcpy(yourname, "Lisa Miller");
strcpy(studentName, yourname);
```

yourname = "Lisa Miller"
studentName = "Lisa Miller"

```
strcmp("Bill", "Lisa");
```

Returns a value < 0

```
strcpy(yourname, "Kathy Brown");
strcpy(myname, "Mark G. Clark");
strcmp(myname, yourname);
```

yourname = "Kathy Brown"
myname = "Mark G. Clark"

Returns a value > 0

Reading/Writing Strings

- most rules that apply to arrays apply to **C**-strings as well.
- Aggregate operations, such as assignment and comparison, are not allowed on arrays.
- We know that the input/output of arrays is done component-wise.
- The one place where C++ allows aggregate operations on arrays is the input and output of **C**-strings (that is, character arrays)

String Input

- `char name[31];`
- `cin >> name;`
- stores the next input C-string into **name**
- The length of the input C-string must be less than or equal to **30**.
- If the length of the input string is **4**, the computer stores the four characters that are input and the null character `'\0'`.
- If the length of the input C-string is more than **30**, then because there is no check on the array index bounds, the computer continues storing the string in whatever memory cells follow **name**.
- This process can cause serious problems, because data in the adjacent memory cells will be corrupted

get Function

- the extraction operator, **>>**, skips all leading whitespace characters and stops reading data into the current variable as soon as it finds the first whitespace character or invalid data
- **C**-strings that contain blanks cannot be read using the extraction operator, **>>**. For example, if a first name and last name are separated by blanks, they cannot be read into **name**.
- `char str[31];`
- `cin.get(str, 31);`
- If the input is:
- **William T. Johnson**
- then **"William T. Johnson"** is stored in **str**. Suppose that the input is:
- **Hello there. My name is Mickey Blair.**
- which is a string of length 37. Because **str** can store, at most, 30 characters, the
- **C**-string **"Hello there. My name is Mickey"** is stored in **str**.

String input cont'd

- `char str1[26];`
- `char str2[26];`
- `char discard;`
- two lines of input:
- **`Summer is warm.`**
- **`Winter will be cold.`**
- suppose that we want to store the first **C**-string in **`str1`** and the second **C**-string in **`str2`**. Both **`str1`** and **`str2`** can store **C**-strings that are up to **`25`** characters in length. Because the number of characters in the first line is **`15`**, the reading stops at **`'\n'`**.

String input cont'd

- Now the newline character remains in the input buffer and must be manually discarded.
- Therefore, you must read and discard the newline character at the end of the first line to store the second line into **`str2`**.
- The following sequence of statements stores the first line into **`str1`** and the second line into **`str2`**:
- **`cin.get(str1, 26);`**
- **`cin.get(discard);`**
- **`cin.get(str2, 26);`**
- Study this for more information:

  https://www.geeksforgeeks.org/clearing-the-input-buffer-in-cc/

getline function

- To read and store a line of input, including whitespace characters, you can also use the stream function **`getline`**. Suppose that you have the following declaration:
- `char textLine[100];`
- `cin.getline(textLine, 100);`
- The above statement will read and store the next 99 characters, or until the newline character, into **`textLine`**. The null character will be automatically appended as the last character of **`textLine`**.

String Output

- **`cout << name;`**
- outputs the contents of **`name`** on the screen.
- The insertion operator, **<<**, continues to write the contents of **`name`** until it finds the null character. Thus, if the length of **`name`** is **4**, the above statement outputs only four characters. If **`name`** does not contain the null character, then you will see strange output because the insertion operator continues to output data from memory adjacent to **`name`** until a **`'\0'`** is found.

Try this code on your computer

```
#include <iostream>

using namespace std;

int main()
{
    char name[5] = {'a', 'b', 'c', 'd', 'e'};
    int x = 50;
    int y = -30;

    cout << name << endl;

    return 0;
}
```

Output:

abcde╠╠╠╠╠╠╠╠σ♥@·I

The `strlen` Function

- `#include <cstring>`
- For instance, the following code segment uses the `strlen` function to determine the length of the string stored in the `name` array:
- `char name[] = "Thomas Edison";`
- `int length;`
- `length = strlen(name);`

The `strcat` Function

- The function *concatenates* or appends one string to another.
- `const int SIZE = 13;`
- `char string1[SIZE] = "Hello ";`
- `char string2[] = "World!";`
- `cout << string1 << endl;`
- `cout << string2 << endl;`
- `strcat(string1, string2);`
- `cout << string1 << endl;`
- These statements will cause the following output:
- `Hello`
- `World!`
- `Hello World!`

The `strcat` Function

Before the call to `strcat (string1, string2)`:

`string1`

H	e	l	l	o		\0						

`string2`

W	o	r	l	d	!	\0

After the call to `strcat (string1, string2)`:

`string1`

H	e	l	l	o		W	o	r	l	d	!	\0

`string2`

W	o	r	l	d	!	\0

The `strncat` and `strncpy` Functions (to avoid the out of bound index)

- the the `strcat` and `strcpy` functions can potentially overwrite the bounds of an array, they make it possible to write unsafe code.
- As an alternative, you should use `strncat` and `strncpy` whenever possible.
- `strncat(string1, string2, 10);`
- When this statement executes, `strncat` will append no more than 10 characters from
- `string2` to `string1` .

Using strncat function

```
1   int maxChars;
2   const int SIZE_1 = 17;
3   const int SIZE_2 = 18;
4
5   char string1[SIZE_1] = "Welcome ";
6   char string2[SIZE_2] = "to North Carolina";
7
8   cout << string1 << endl;
9   cout << string2 << endl;
10  maxChars = sizeof(string1) - (strlen(string1) + 1);
11  strncat(string1, string2, maxChars);
12  cout << string1 << endl;
```

The statement in line 10 calculates the number of empty elements in `string1` . It does this by subtracting the length of the string stored in the array plus 1 for the null terminator. This code will cause the following output:

```
Welcome
to North Carolina
Welcome to North
```

Using strncpy

- `strncpy(string1, string2, 5);`
- When this statement executes, `strncpy` will copy no more than five characters from
- `string2` to `string1` .

Using strncpy

```
1  int maxChars;
2  const int SIZE = 11;
3
4  char string1[SIZE];
5  char string2[] = "I love C++ programming!";

7   maxChars = sizeof(string1) - 1;
8   strncpy(string1, string2, maxChars);
9   // Put the null terminator at the end.
10  string1[maxChars] = '\0';
11  cout << string1 << endl;
```

Exercises

- Find String length, Compare strings,
- Find substring and replace,
- Calculate frequency of specific characters
- Remove specific characters.
- Detecting Palindromes

Reading assignment

- **Specifying Input/Output Files at Execution Time (DS Malik page 559)**
- **`string` Type and Input/Output Files (DS Malik page 559)**
- **`string` Type (DS Malik page 492). Explore more about the C++ String type in this section, you can also find a lot of interesting pre-defined functions of string type in this section (but keep in mind the difference of string and c-string).**

References

1. C++ Programming: From Problem Analysis to Program Design, Third Edition
2. https://www.just.edu.jo/~yahya-t/cs115/