Strings

Up until now we have only used **literal strings**, characters surrounded by double quotes, such as “Hello World!” We haven’t saved any of those strings as variables. C and Java have very different ways of storing strings: we will start with C, which does not have a built-in string type, and then cover Java, which does. First, however, we are going to talk about the characters that make up strings.

# Characters

Character types are actually just integers. The char variable in C is explicitly an integer type, holding numbers from -128 to 127. These numbers can then be interpreted many different ways: we will be using the ASCII character encoding standard.

When we use chars to store letters and numbers, what they are actually storing is the ASCII code for that particular character. An uppercase A, for example, is 65. We tell the compiler that we intend something to be a character by using single quotes.

char myChar;

myChar = 'A'; /\* myChar's data is the number 65 \*/

myChar = '8'; /\* the ASCII for 8 is 56, so myChar is now 56 \*/

In C, we can set a char’s value with an integer, such as saying character = 65 to set it to a capital A, but it is rarely a good idea to use ASCII codes directly. This would make your code less readable and less flexible.

Because char variables are actually integers, we can do math with them.

char myChar = 'A'; /\* myChar is 65 \*/

myChar = 'A' + 2; /\* myChar is now 67, or C \*/

This is also useful when we want to check if a character is within a certain range, such as whether it is a number or a lowercase letter.

if (myChar >= '0' && myChar <= '9')

{

/\* condition will be true if myChar is a number \*/

}

else if (myChar >= 'a' && myChar <= 'z')

{

/\* condition will be true if myChar is a lowercase letter \*/

}

## Escape Sequences

There are some special character combinations that we have to use for certain characters or behaviors in strings. We have already seen the newline character, \n. Similarly, there is \t to insert a tab.

Imagine trying to print this phrase: Enter the word “hello”.

printf("Enter the word "hello".");

Do you see the problem? The compiler doesn’t know that the double quote before the word “hello” is not the end of the string. To use double and single quotes as characters, we have to use the escape sequences \" and \'. And, since a backslash denotes an escape sequence, to use a backslash as a character we have to use \\.

printf("Enter the word \"hello\".");

# Strings in C

As mentioned above, C does not have a built-in string type. Instead, char arrays are used to implement strings. Since there is no way to tell from an array variable how long it is, the last character in a C string is always integer zero, or the escape sequence \0, also called the **null** character. This is how we can find the end of the string. Because of this C-style strings are also commonly called null-terminated strings.

The following example creates a string variable and uses the printf() function to print the string to the console:

char myString[] = "Hello World!";

printf("%s", myString);

With the console output being:

Hello World!

The myString array above is created as an array of 13 chars, as shown below.

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

When we initialize a string using a string literal, text within double quotes, we don’t need to add the null character at the end. However, if we create the string manually, with an initializer list or by overwriting the string once it’s created, we must remember to add the null character ourselves.

There are two ways to insert a null character into a string: we can either use the integer zero or the null character '\0'. We can’t use the character zero, '0', because its integer value is not actually zero (it is 48 in ASCII) so it does not terminate the string.

char myString[] = { 'H', 'e', 'l', 'l', 'o' };

printf("%s", myString); /\* **error**: the string has no null termination \*/

char myString[10] = { 'H', 'e', 'l', 'l', 'o' };

/\* This will print correctly since all following characters after the initialization are set to zero \*/

myString[5] = 0; /\* correct, last character of string is integer zero \*/

myString[5] = '\0'; /\* also correct, uses ASCII equivalent of zero \*/

myString[5] = '0'; /\* incorrect, the character is not integer zero \*/

This char array can be accessed just like any other array. myString[4] would give us the character ‘o’. If we wanted to print the string the hard way by iterating through every character, we can find when to stop by looking for either zero or the null character.

char myString[] = "Hello World!";

int i;

for (i = 0; myString[i] != 0; ++i)

{

printf("%c", myString[i]);

}

for (i = 0; myString[i] != '\0'; ++i)

{

printf("%c", myString[i]);

}

## C String Output and Input

A convenient function for printing strings is puts(). This function prints a string to the console and automatically adds a newline after it.

|  |  |
| --- | --- |
| **Code** | **Output** |
| char myString[] = "Hello";  puts(myString);  puts("World"); | Hello  World |

There is also a function called putchar() to print a single character (this one does not insert a newline):

|  |  |
| --- | --- |
| **Code** | **Output** |
| char myString[] = "Hello";  putchar(myString[0]);  putchar('i'); | Hi |

To get input from the user we can use the function gets(). This function will take a line of input from the user and put it into the supplied char array with a null termination.

char input[100];

puts("Enter some input: ");

gets(input);

You may notice a potential problem here. We have created an array of 100 characters, but the gets() function has no way of knowing the size of that array. If the user enters more than 99 characters, the data will be larger than the array, and other memory will be overwritten.

A safer alternative is the fgets() function. This function takes a char array, an int with the maximum number of characters to read, and a third parameter tells it where to read from. (The function is designed to read from files as well as from user input). To tell it to read input from the console we pass in stdin for the standard input stream.

#define MAX\_INPUT 20

char input[MAX\_INPUT];

puts("Enter some input: ");

fgets(input, MAX\_INPUT, stdin);

There is also an input function to get a single character, getchar(). We don’t pass anything into this function: instead it returns the character that it reads.

char character;

puts("Enter a single character: ");

character = getchar();

## C String Functions

There are several functions that are made specifically for null-terminated strings. You will need to include the string.h header in order to use them.

#include <string.h>

|  |  |
| --- | --- |
| **Function Prototype** | **Description** |
| size\_t strlen(char[] string); | Returns the length of the string, which is the number of characters int the string. It does not include the terminating 0. |
| char[] strcpy(char[] destination,  char[] source); | Copies the string pointed to by source into the string pointed to by destination. Destination must have enough space to hold the string from source. The return is destination. |
| char[] strcat(char[] destination,  char[] source); | Concatenates (joins) two strings by appending the string in source to the end of the string in destination. Destination must have enough space to accommodate both strings. The return is destination. |
| int strcmp(char[] string1, char[]  string2); | Compares two strings lexicographically (i.e. alphabetically). If string1 is less than string2, the return value is negative. If string1 is greater than string2, then the return value is positive. Otherwise the return is 0 (they are the same.) |

# Strings in Java

Unlike C, Java does have a built-in string object. There are two ways to create a String, either with the String object’s constructor or with a string literal.

String string1 = new String("Using the String constructor");

String string2 = "Using a string literal";

Both of these methods have the same result.

Once a String object is created, it can’t be changed. Whichever method you use to create the String, the text you create it with is the text it will always have.

The String object can be passed into the print functions instead of a string literal.

System.out.println(string1);

You might remember from the chapter section on console output in Java that we were able to combine string literals and numbers in the print function, like this:

System.out.println("Hello " + 2 + " the World");

Output:

Hello 2 the World

We can also combine, or concatenate, String objects the same way.

String string1 = "Hello";

String string2 = "World";

System.out.println(string1 + " " + string2);

Output:

Hello World

We can also concatenate string literals and variables when creating a String object.

int number = 5;

String myString = "I have " + number + " objects.";

System.out.println(myString);

Output:

I have 5 objects.

## Java String Functions

There are many built-in String functions in Java, some of which we describe below. Feel free to experiment with the rest on your own. These functions are all called on the String object using the dot operator, like this:

String myString = "Hello";

char firstChar = myString.charAt(0);

|  |  |
| --- | --- |
| **Function Prototype** | **Description** |
| char charAt(int index); | Returns the character value at a specified index in the String, 0 being the first character. |
| int length(); | Returns the length of the string (the number of characters in the string). |
| String toLowerCase(); | Returns a new String where every uppercase letter has been replaced by its lowercase equivalent. The original String is unchanged. |
| String toUpperCase(); | Returns a new String where every lowercase character has been replaced by its uppercase equivalent. The original String is unchanged. |
| boolean equals(String str); | Returns true if both Strings contain the same character sequence and false if they do not. |
| int indexOf(String str); | Returns the index of the first occurrence of a given string (str) in the String object. Returns -1 if str is not found. Note that this method is case-sensitive. |
| int indexOf(String str, int index); | Returns the index of the first occurrence of a given string (str) in the String object starting at a specified index in the string. Returns -1 if str is not found. Note that this method is case-sensitive. |
| String substring(int begin); | Returns a new String object containing the characters from the begin index until the end of the String. |
| String substring(int begin, int  end); | Returns a new String object containing the characters from the begin index until the end index (not including the end index). |
| String replace(char oldChar, char  newChar); | Returns a new String containing the old String with all instances of oldChar replaced with newChar. The original String is unchanged. |
| String replace(CharSequence target,  CharSequence replacement); | Returns a new String containing the old String with all instances of target replaced with replacement. The original String is unchanged.  Note: In this context a CharSequence is simply a String. |