Chapter 6 :- Strings

Syntax and use string function

Function	What It Does
strcmp()	Compares two strings in a case-sensitive way. If the strings match, the function returns 0.
strncmp()	Compares the first n characters of two strings, returning 0 if the given number of characters match.
strcasecmp()	Compares two strings, ignoring case differences. If the strings match, the function returns 0.
strncasecmp()	Compares a specific number of characters between two strings, ignoring case differences. If the number of characters match, the function returns 0.

Function	What It Does
strcat()	Appends one string to another, creating a single string out of two.
strncat()	Appends a given number of characters from one string to the end of another.
strchr()	Searches for a character within a string. The function returns that character's position from the start of the string as a pointer.
strrchr()	Searches for a character within a string, but in reverse. The function returns the character's position from the end of the string as a pointer.
strstr()	Searches for one string inside another string. The function returns a pointer to the string's location if it's found.

Function	What It Does
strnstr()	Searches for one string within the first n characters of the second string. The function returns a pointer to the string's location if it's found.
strcpy()	Copies (duplicates) one string to another.
strncpy()	Copies a specific number of characters from one string to another.
strlen()	Returns the length of a string, not counting the or NULL character at the end of the string.

Array of Strings

A string is a 1-D array of characters, so an array of strings is a 2-D array of characters.

Just like we can create a 2-D array of int, float etc; we can also create a 2-D array of character or array of strings.

```
char ch_arr[3][10] = {
{'s','p','i','k','e', '\0'},
{'t', 'o', 'm','\0'},
{'j', 'e', 'r', 'r', 'y','\0'}
 };
```

It is important to end each 1-D array by the null character, otherwise, it will be just an array of characters. We can't use them as strings. Declaring an array of strings this way is rather tedious, that's why C provides an alternative syntax to achieve the same thing.

This above initialization is equivalent to:

```
char ch_arr[3][10] = {"spike", "tom", "jerry"};
```

The first subscript of the array i.e 3 denotes the number of strings in the array and the second subscript denotes the maximum length of the string. Recall the that in C, each character occupies 1 byte of data, so when the compiler sees the above statement it allocates 30 bytes (3*10) of memory.

The following program demonstrates how to print an array of strings.

```
#include<stdio.h>
int main()
{
int i;
char ch_arr[3][10] = {"spike", "tom","jerry"};
printf("1st way \n\n");
    for(i = 0; i < 3; i++)
    {
        printf("string = %s \t address = %u\n", ch_arr + i, ch_arr + i);
    }
```

```
 // signal to operating system program ran fine
    return 0;
}
```

Expected Output:

```
string = spike address = 2686736
string = tom address = 2686746
string = jerry address = 2686756
```

Strings & Pointer

Creating a pointer for the string

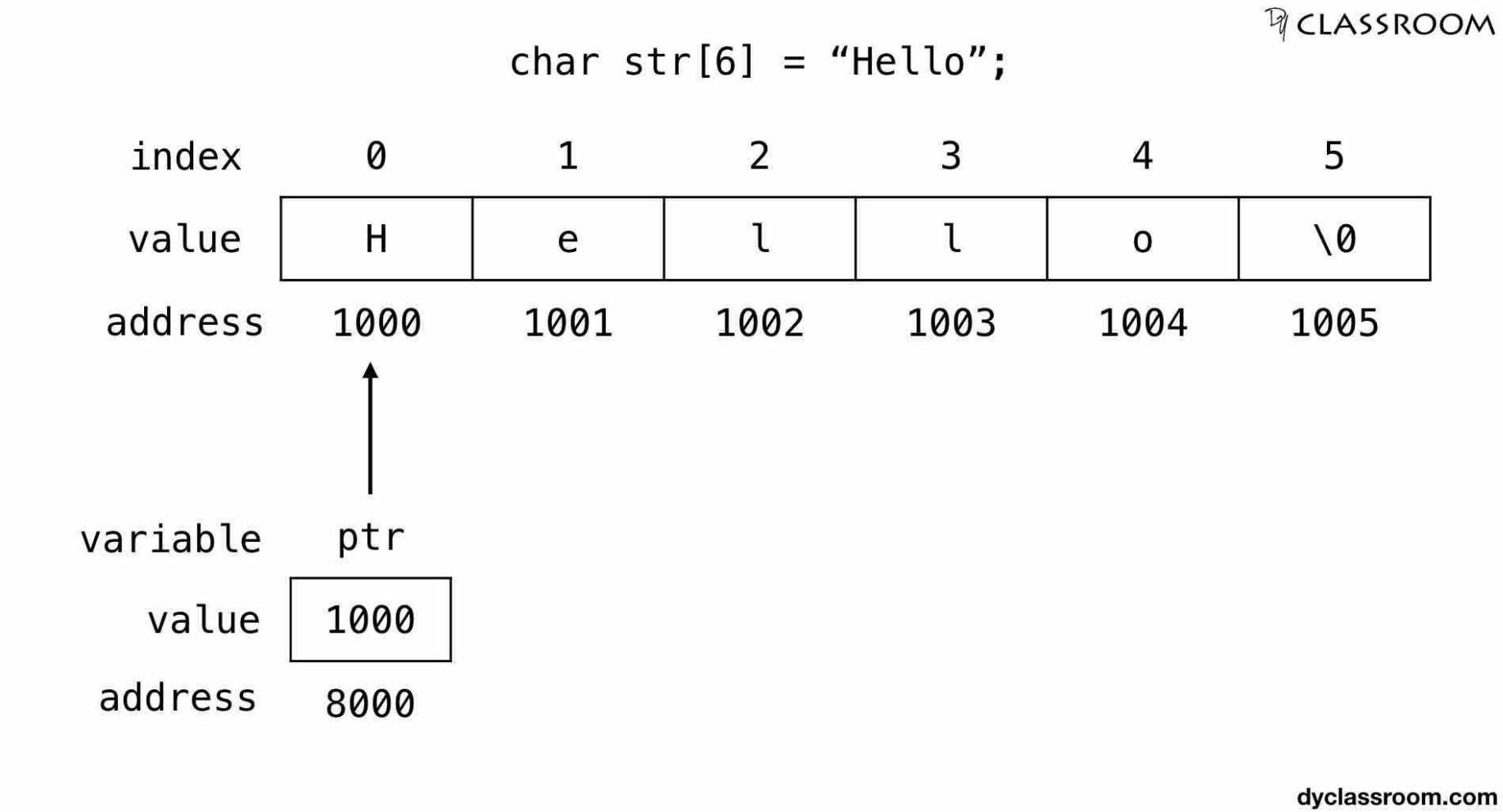
The variable name of the string str holds the address of the first element of the array i.e., it points at the starting memory address. So, we can create a character pointer ptr and store the address of the string str variable in it. This way, ptr will point at the string str.

In the following code we are assigning the address of the string str to the pointer ptr.

```
char *ptr = str;
```

We can represent the character pointer variable ptr as follows.

The pointer variable ptr is allocated memory address 8000 and it holds the address of the string variable str i.e., 1000.



Accessing string via pointer

To access and print the elements of the string we can use a loop and check for the \0 null character. In the following example we are using while loop to print the characters of the string variable str.

```
#include <stdio.h>
int main(void) {
  // string variable
  char str[6] = "Hello";
 // pointer variable
  char *ptr = str;
```

```
  // print the string
  while(*ptr != '\0') {
printf("%c", *ptr);
  // move the ptr pointer to the next memory location
    ptr++;
  }  return 0;
}
```

********** END OF PROGRAM **********

Using pointer to store string

We can achieve the same result by creating a character pointer that points at a string value stored at some memory location. In the following example we are using character pointer variable strPtr to store string value.

```
#include <stdio.h>
int main(void) {
// pointer variable to store string
 char *strPtr = "Hello";
// temporary pointer variable
```

```
 char *t = strPtr;
// print the string
  while(*t != '\0') {
   printf("%c", *t);
// move the t pointer to the next memory location
    t++;
  }  return 0;
}
```

********** END OF PROGRAM **********

Command Line Arguments

The arguments passed from command line are called command line arguments. These arguments are handled by main() function. To support command line argument, you need to change the structure of main() function as given below.

int main(int argc, char *argv[])

Here, argc counts the number of arguments. It counts the file name as the first argument.

The argv[] contains the total number of arguments. The first argument is the file name always.

Let's see the example of command line arguments where we are passing one argument with file name.

```
#include <stdio.h>
void main(int argc, char *argv[] )  {
printf("Program name is: %s\n", argv[0]);
if(argc < 2){
printf("No argument passed through command line.\n");  }
else{
printf("First argument is: %s\n", argv[1]);
   }
}
```