String Library

Learning Outcomes

After reading this section, you will be able to:

- Implement algorithms using standard library procedures to incorporate existing technology
- Stream data using standard library functions to interact with users

Introduction

The standard library that ships with C compilers and processes character strings is called the string library. The functions in this library perform fundamental operations on character strings, which include copying one string to another, adding one string to another and comparing one string to another.

This chapter describes four library functions that operate on character strings: determining the length, copying one to another, comparing one to another and concatenating one to another.

String Functions

The functions of the string library include:

- `strlen()` - returns the number of characters in a character string
- `strcpy()` - copies one character string to another
- `strcmp()` - compares one character string to another
- `strcat()` - concatenates one character string to another

> **NOTE**
>
> These are four of the most common string functions, but there are many available in the string library.

The header file that contains the prototypes for these library functions is:

```
#include <string.h>
```

String Length

The `strlen()` function returns the number of characters in the character string excluding the **null terminator byte**. That is, `strlen()` returns the index of the **null terminator byte**.

The following program finds the length of the input string and reverses its contents:

```
// Reverse a string
// reverse_string.c

#include <stdio.h>
#include <string.h>

int main(void)
{
    int i, len;
    char str[31], rev[31];

    printf("Enter a string   : ");
    scanf("%30[^\n]%*c", str);

    printf("In reverse order : ");
    len = strlen(str);

    for (i = len - 1; i >= 0; i--)
    {
        rev[len - 1 - i] = str[i];
    }

    rev[len] = '\0';
    puts(rev);

    return 0;
}
```

The above program produces the following output:

```
Enter a string   : strlen
In reverse order : nelrts
```

String Copy

The `strcpy()` function receives two addresses and copies the string at the second address into the memory locations starting at the first address. `strcpy()` returns the address of the destination string.

We are responsible for ensuring that there is sufficient room in the destination string to hold all of the characters in the source string **including the null terminator byte**.

The following program copies the input string only if it contains 20 characters or less:

```c
// Copy a string
// copy_string.c

#include <stdio.h>
#include <string.h>

int main(void)
{
    char str[31], copy[21] = "?";
    int i, len;

    printf("Source : ");
    scanf("%30[^\n]%*c", str);

    len = strlen(str);

    if (len < 21)
    {
        strcpy(copy, str);
        printf("Copy : %s\n", copy);
    }
    else
    {
        printf("* No room *\n");
        printf("Copy : %s\n", copy);
    }
```

```
    return 0;
}
```

Sample run-through #1 of the above program:

```
Source : strcpy
Copy   : strcpy
```

Sample run-through #2 of the above program:

```
Source : this string is too long
* No room *
Copy   : ?
```

String Compare

The `strcmp()` function receives two addresses and compares the string at the first address to the string at the second address.

`strcmp()` returns:

- **0** if they are identical
- **a negative value** if the first non-matching character in the first string is, under the host computer's collating sequence, **lower** than the character with the same index in the second string
- **a positive value** if the first non-matching character in the first string is, under the host computer's collating sequence, **higher** than the character with the same index in the second string

For example:

```
// Compare Two Strings
// compare_string.c

#include <stdio.h>
#include <string.h>
```

```c
int main(void)
{
    char str1[31], str2[31];
    int i, len;

    printf("Enter first string  : ");
    scanf("%30[^\n]%*c", str1);

    printf("Enter second string : ");
    scanf("%30[^\n]%*c", str2);

    if (strcmp(str1, str2) < 0)
    {
        printf("%s precedes %s\n", str1, str2);
    }
    else if(strcmp(str1, str2) > 0)
    {
        printf("%s follows %s\n", str1, str2);
    }
    else
    {
        printf("%s matches %s\n", str1, str2);
    }

    return 0;
}
```

Sample run-through #1 of the above program:

```
Enter first string  : elephant
Enter second string : elephants
elephant precedes elephants
```

Sample run-through #2 of the above program:

```
Enter first string  : elephant
Enter second string : elephant
elephant matches elephant
```

Sample run-through #3 of the above program:

```
Enter first string  : elephants
Enter second string : elephant
elephants follows elephant
```

String Concatenate

The `strcat()` function receives two addresses and concatenates the string at the second address to the string at the first address. `strcat()` returns the address of the concatenated string.

We are responsible for ensuring that the destination string has enough room to hold the concatenated result **along with the null terminator byte**.

For example:

```c
// Concatenate two strings
// concatenate.c

#include <stdio.h>
#include <string.h>

int main(void)
{
    int i, len;
    char surname[31], fullName[62];

    printf("Given name  : ");
    scanf("%30[^\n]%*c", fullName);

    printf("Surname     : ");
    scanf("%30[^\n]%*c", surname);

    strcat(fullName, " ");
    strcat(fullName, surname);
    printf("Full name is %s\n", fullName);

    return 0;
}
```

Sample run-through of the above program:

```
Given name  : Jane
Surname     : Doe
Full name is Jane Doe
```

> **NOTE**
>
> We have allocated 62 characters to accommodate 30+30 characters plus the blank space separator and the null terminator byte.