

UESTC1005 - Introductory Programming

Strings & Pointers (Intro)

Week 10 | Lecture 9

Dr Ahmed Zoha

Lecturer in Communication Systems

School of Engineering

University of Glasgow

Email: ahmed.zoha@Glasgow.ac.uk

Example (RECAP)

- Let's look at the 'Get name' example program:

```
int main()
{
    char string1[ 20 ], string2[] = "string literal";
    int i;

    printf(" Enter a string: ");
    scanf( "%s", string1 );
    printf( "string1 is: %s\nstring2: is %s\n"
           "string1 with spaces between characters is:\n",
           string1, string2 );

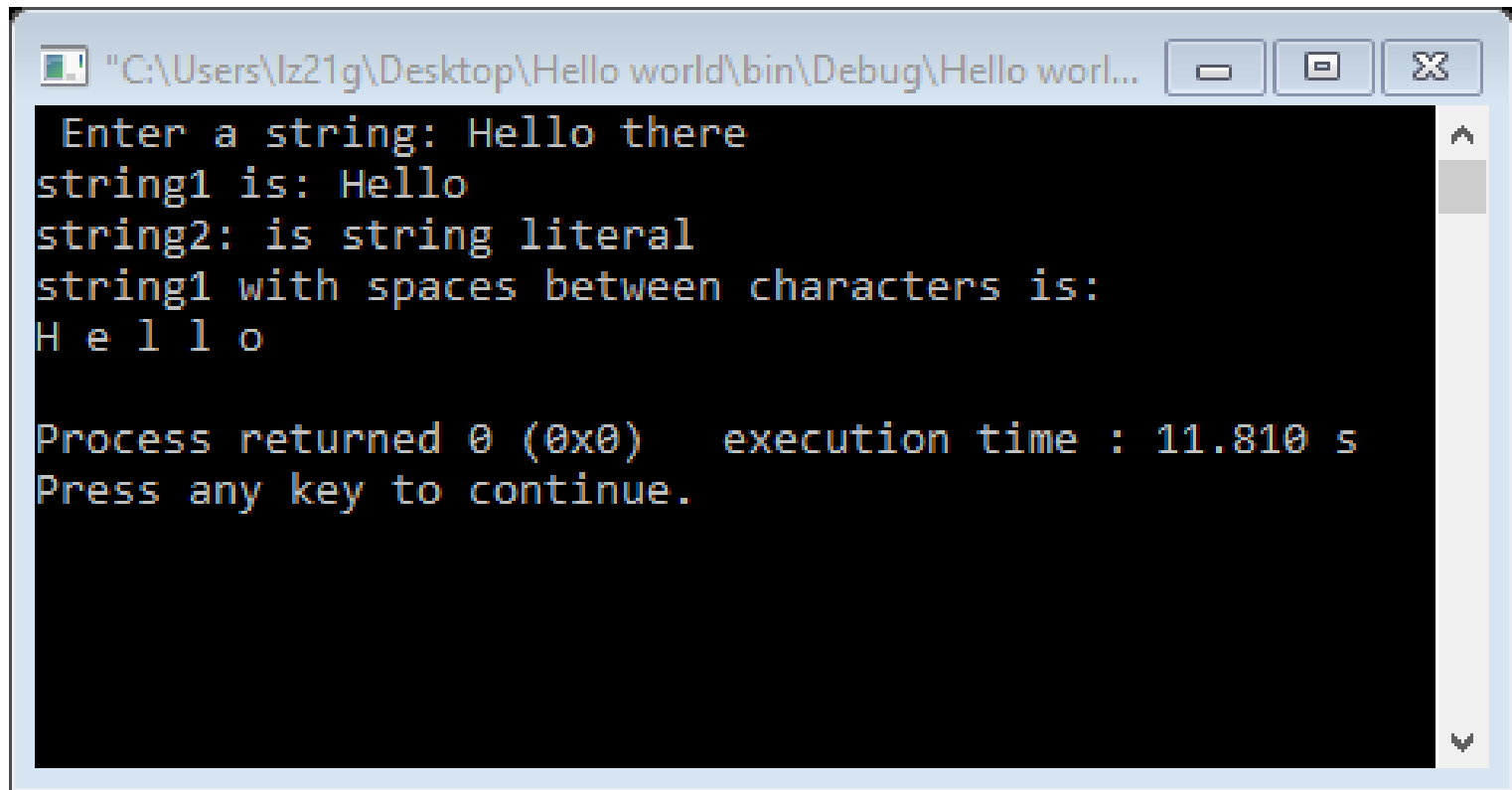
    for ( i = 0; string1[ i ] != '\0'; i++ )
        printf( "%c ", string1[ i ] );

    printf( "\n" );
    return 0;
}
```

- Here we can see that the *scanf* function stores the users input into a string variable called string1
- The format specifier *%s* means that you can read a string from the user
- Notice that the *&* symbol is not needed in this case as it was when we used *scanf* earlier
- The reasons for this will be come clear when pointers are covered

Example

- When the above code is compiled and executed, it produces the following result:



```
"C:\Users\lz21g\Desktop\Hello world\bin\Debug\Hello worl...  
Enter a string: Hello there  
string1 is: Hello  
string2: is string literal  
string1 with spaces between characters is:  
H e l l o  
  
Process returned 0 (0x0)   execution time : 11.810 s  
Press any key to continue.
```

Problem??

- Why the output shows only **Hello**, even though I have entered Hello there.
- The reason is that if we read a string by using **"%s" format specifier**, string will be terminated when white space is found.



How to read string with spaces in C?

- Read string with spaces by using **"%[^\n]" format specifier**
- The format specifier **"%[^\n]"** tells to the compiler that read the characters until "\n" (end-of-line) is not found.

Example 1 (Read String with Spaces using **scanf**)

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  int main()
6  {
7      char string1[20], string2[] = "string literal";
8      int i;
9
10     printf("Enter a String:");
11     scanf("%[^\n]", string1);
12     printf("string1 is: %s\nstring2 is %s\n"
13           "string1 with spaces between characters is :\n", string1, string2);
14     for (i=0; string1[i] != '\0'; i++)
15     {
16         printf("%2c", string1[i]);
17     }
18     printf("\n");
19
20     return 0;
21 }
```

"C:\Zoha\Work\Teaching\2019\Course_IP_Sep2019_Dec2019\Code\Sting Print\bin\Debug\Sting Print.exe"

```
Enter a String:Hello There
string1 is: Hello There
string2 is string literal
string1 with spaces between characters is :
H e l l o   T h e r e
```

```
Process returned 0 (0x0)   execution time : 7.615 s
Press any key to continue.
```

Ex 2- Count the number of word in a string input by the user

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include<string.h>
4
5  void main()
6  {
7      char string[200];
8      int count=0, i;
9
10     printf("Enter the string:\n");
11     scanf("%[^\n]s",string);
12
13     for(i=0;string[i]!='\0';i++)
14     {
15         if(string[i]==' ' && string[i+1]!=' ')
16         {
17             count++;
18         }
19     }
20
21     printf("The total number of words in the input strings are: %d",count+1);
22 }
23
```

C:\Zoha\Work\Teaching\2019\Course_IP_Sep2019_Dec2019\Code\Ex2_L9_WordC

```
Enter the string:
How are you doing today
The total number of words in the input strings are: 5
Process returned 53 (0x35)   execution time : 27.031 s
Press any key to continue.
```

Recap- Defining Constant Strings

- The pre-processor directive `#define` let you define constant string which makes the program more readable.
- Remember we have used constants in the weather program (L8)
 - `#define MONTHS 12`
 - `#define YEARS 5`
- Important things to remember
 - No semicolon at the end
 - No equal sign used to assign values to string
 - There is no such thing as local define since they can appear anywhere in the program
 - String Messages
 - `#define Message "You have won the game!"`

String Functions

- Commonly Performed operations on character strings include
 - Getting the length of the string
 - **strlen()**
 - Copying one character string to another
 - **strcpy()** and **strncpy()**
 - Combining two character strings together (concatenation)
 - **strcat()** and **strncat()**
 - Determining if two character strings are equal
 - **strcmp()** and **strncmp()**
 - **All these functions are defined in the `string.h` header file**

Getting the length of the string (1/2)

- The `strlen()` function finds the length of the string
- It doesn't count null character `'\0'`.
- How about length of character arrays?

```
1  #include<stdio.h>
2  #include <string.h>
3
4  int main()
5  {
6      char ch[]={'g', 'e', 'e', 'k', 's', '\0'};
7
8      printf("Length of string is: %d", strlen(ch));
9
10     return 0;
11 }
12
```

- **Output: The length of the string is 5**

Getting the length of the string (2/2)

```
1 // c program to demonstrate
2 // example of strlen() function.
3
4 #include<stdio.h>
5 #include <string.h>
6
7 int main()
8 {
9     char str[] = "geeks";
10
11     printf("Length of string is: %d", strlen(str));
12
13     return 0;
14 }
15
```

Output: The length of the string is 5

Copying Strings (1/2)

```
char s[100]; //declare  
S = "hello"; // Does not work "lvalue required" error
```

- Direct assignment of string or character arrays using = sign is not possible.
- You have to use **strcpy()** function to copy an existing string

```
char src[50], dest[50];  
strcpy(src, "This is source");  
strcpy(dest, "This is destination");
```

- The **strcpy()** function does not check to see whether the source string actually fits in the target string

Copying Strings (2/2)

- Safer way to copy strings is to use **strncpy()**
- **strncpy()** takes a third argument (the maximum number of characters to copy)

```
1  /* Fig. 8.18: fig08_18.c
2     Using strcpy and strncpy */
3  #include <stdio.h>
4  #include <string.h>
5
6  int main()
7  {
8     char x[] = "Happy Birthday to You"; /* initialize char array x */
9     char y[ 25 ]; /* create char array y */
10    char z[ 15 ]; /* create char array z */
11
12    /* copy contents of x into y */
13    printf( "%s%s\n%s%s\n",
14           "The string in array x is: ", x,
15           "The string in array y is: ", strcpy( y, x ) );
16
17    /* copy first 14 characters of x into z. Does not copy null
18       character */
19    strncpy( z, x, 14 );
20
21    z[ 14 ] = '\0'; /* terminate string in z */
22    printf( "The string in array z is: %s\n", z );
23
24    return 0; /* indicates successful termination */
25 } /* end main */
```

- Note strncpy() does not copy a null and you have to manually Insert it.
- Alternatively you can use `memset(z, '\0', sizeof(z));`
- Output of the program is

```
The string in array x is: Happy Birthday to You
The string in array y is: Happy Birthday to You
The string in array z is: Happy Birthday
```

String Concatenation

- The **strcat()** function takes two strings for arguments
 - A copy of the second string is concatenated to the end of first
 - The combined version becomes the new first string (i.e., s1)
 - The second string is not altered (i.e., s2)

```
3  #include <stdio.h>
4  #include <string.h>
5
6  int main()
7  {
8      char s1[ 20 ] = "Happy "; /* initialize char array s1 */
9      char s2[] = "New Year "; /* initialize char array s2 */
10     char s3[ 40 ] = "";      /* initialize char array s3 to empty */
11
12     printf( "s1 = %s\ns2 = %s\n", s1, s2 );
13
14     /* concatenate s2 to s1 */
15     printf( "strcat( s1, s2 ) = %s\n", strcat( s1, s2 ) );
16
17     /* concatenate first 6 characters of s1 to s3. Place '\0'
18        after last character */
19     printf( "strncat( s3, s1, 6 ) = %s\n", strncat( s3, s1, 6 ) );
20
21     /* concatenate s1 to s3 */
22     printf( "strcat( s3, s1 ) = %s\n", strcat( s3, s1 ) );
23
24     return 0; /* indicates successful termination */
25 } /* end main */
26
27
```

The **strcat()** function appends a copy of the string pointed to by s2 (including the terminating null character)

Note the use of **strncat()**

The output of the program is

```
s1 = Happy
s2 = New Year
strcat( s1, s2 ) = Happy New Year
strncat( s3, s1, 6 ) = Happy
strcat( s3, s1 ) = Happy Happy New Year
```

Comparing Strings (1/3)

- Suppose you want to compare someone's response to a stored string
 - You cannot use `==` since this will only check to see if the string has the same address
- There is a function that compares string contents not their addresses
 - **`strcmp()`** is used for string comparison
 - It does not compares characters, so for example you can compare arguments such as “Apples” and “A”, but you can use character argument such as ‘A’.
- This function does for string what relational operators do for numbers
 - It returns 0 if two arguments are the same and nonzero otherwise

Comparing Strings (2/3)

- It returns value < 0 when str 1 is less than str 2
- It returns value > 0 when str 1 is greater than str 2

```
1  #include <stdio.h>
2  #include <string.h>
3  int main () {
4      char str1[20];
5      char str2[20];
6      int result;
7
8      //Assigning the value to the string str1
9      strcpy(str1, "hello");
10
11     //Assigning the value to the string str2
12     strcpy(str2, "hEllo");
13
14     result = strcmp(str1, str2);
15
16     if(result > 0) {
17         printf("ASCII value of first unmatched character of str1 is greater than str2");
18     } else if(result < 0) {
19         printf("ASCII value of first unmatched character of str1 is less than str2");
20     } else {
21         printf("Both the strings str1 and str2 are equal");
22     }
23     return 0;
24 }
25
```

ASCII value of first unmatched character of str1 is greater than str2

The ASCII value of the first unmatched character 'e' is 101 which is greater than ASCII value of 'E' that is 69

Comparing Strings (3/3)

strncmp() on the other hand compares up to n characters of string s1 to s2

```
3  #include <stdio.h>
4  #include <string.h>
5
6  int main()
7  {
8      const char *s1 = "Happy New Year"; /* initialize char pointer */
9      const char *s2 = "Happy New Year"; /* initialize char pointer */
10     const char *s3 = "Happy Holidays"; /* initialize char pointer */
11
12     printf("%s%s\n%s%s\n\n%s%2d\n%s%2d\n%s%2d\n\n",
13           "s1 = ", s1, "s2 = ", s2, "s3 = ", s3,
14           "strcmp(s1, s2) = ", strcmp( s1, s2 ),
15           "strcmp(s1, s3) = ", strcmp( s1, s3 ),
16           "strcmp(s3, s1) = ", strcmp( s3, s1 ) );
17
18     printf("%s%2d\n%s%2d\n%s%2d\n",
19           "strncmp(s1, s3, 6) = ", strncmp( s1, s3, 6 ),
20           "strncmp(s1, s3, 7) = ", strncmp( s1, s3, 7 ),
21           "strncmp(s3, s1, 7) = ", strncmp( s3, s1, 7 ) );
22
23     return 0; /* indicates successful termination */
24
25 }
```

s1 = Happy New Year
s2 = Happy New Year
s3 = Happy Holidays

strcmp(s1, s2) = 0
strcmp(s1, s3) = 1
strcmp(s3, s1) = -1

strncmp(s1, s3, 6) = 0
strncmp(s1, s3, 7) = 1
strncmp(s3, s1, 7) = -1

Comparing Strings

- The **strcmp()** compares the strings until it finds the corresponding characters that differ
 - Could take the search to end of one of the strings.
- **strncmp()** compares the strings until they differ or until it has compared a number of characters specified by a third argument
 - This is normally good for finding prefixes

```
if (strcmp("astronomy","astro",5) == 0)
{
    printf("Found: astronomy");
}
```

Search, tokenize and Analyze Strings

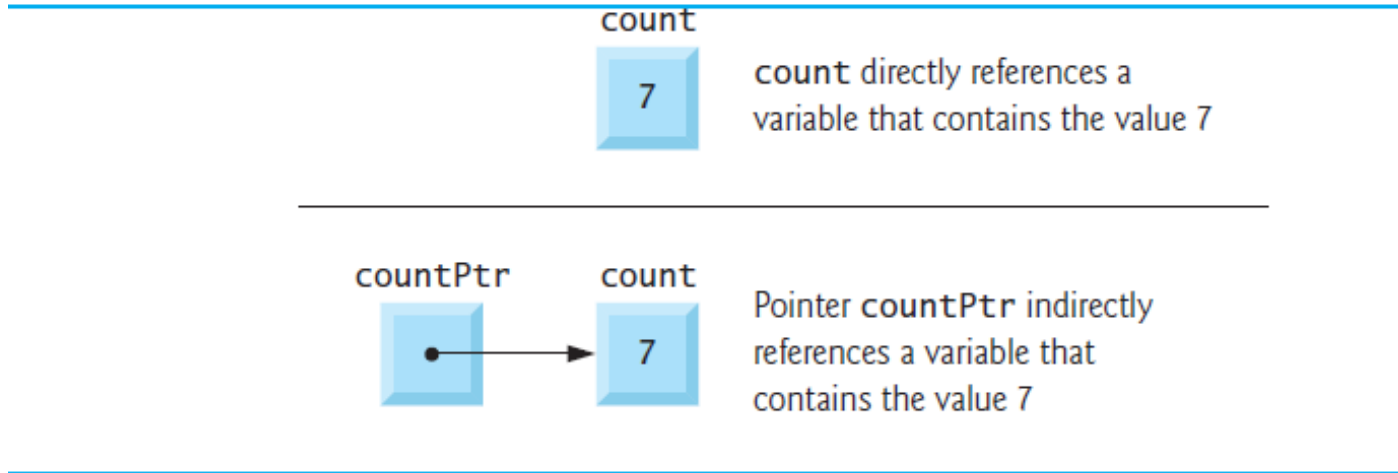
- Searching a string
 - string.h header file declares several string searching functions for finding a single character or a substring
 - strchr() and strstr()
- Tokenize a string
 - A token is a sequence of characters within a string that is bounded by a delimiter (space, comma, etc)
 - The process of breaking a sentence into word according to a specified delimiter is called tokenization
 - strtok()
- Analyzing strings
 - islower(), isupper(), isalpha(), isdigit() etc.

To understand these concept better we will briefly introduce about Pointers

Concept of Pointers

- Pointers the most powerful features of C programming language
- Pointers are variables whose values are memory addresses
 - A variable directly contains a specific value
 - A pointer contains an address of a variable
 - Example of scanf() function that takes address of the variable using &

Pointer Variable Definitions



- A variable name directly references a value, and a pointer indirectly references a value.
- Referencing a value through a pointer is called indirection.

Pointer Variable Definitions

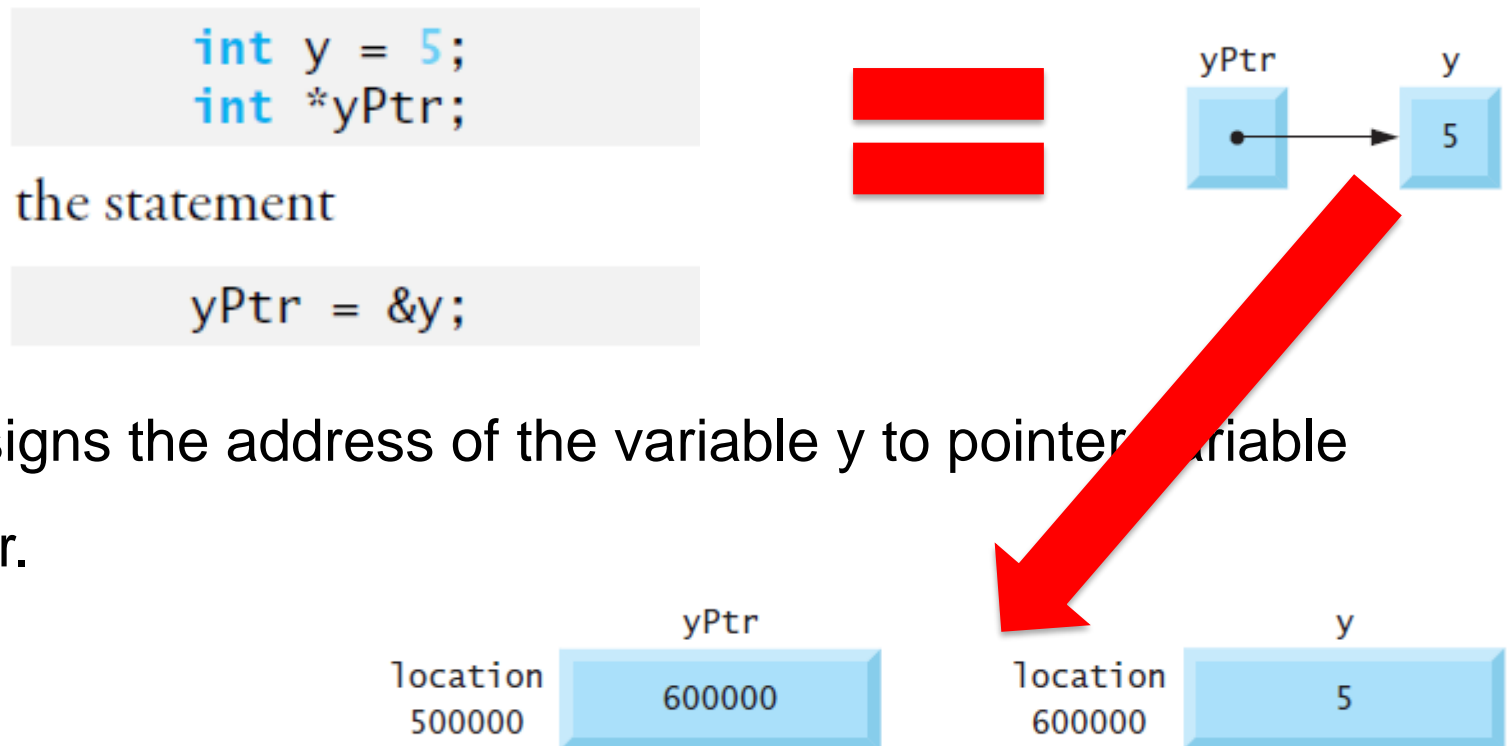
- Pointers, like all variables, must be defined before they can be used. For example:

```
int *countPtr;
```

- Specifies that variable countPtr is of type int * (i.e., a pointer to an integer)
- It is read: “countPtr is a pointer to int” or “countPtr points to an object of type int.”

Pointer Operators

- The `&`, or address operator, is a unary operator that returns the address of its operand. E.g.,



- assigns the address of the variable `y` to pointer variable `yPtr`.

Pointer Operators

- The unary `*` operator referred to as the indirection operator or dereferencing operator, returns the value of the object to which its operand (i.e., a pointer) points. E.g.,
`printf("%d", *yPtr);`
- prints the value of variable `y`, namely 5.

Example

```
int main()
{
    int a;           /* a is an integer */
    int *aPtr;       /* aPtr is a pointer to an integer */

    a = 7;
    aPtr = &a;       /* aPtr set to address of a */

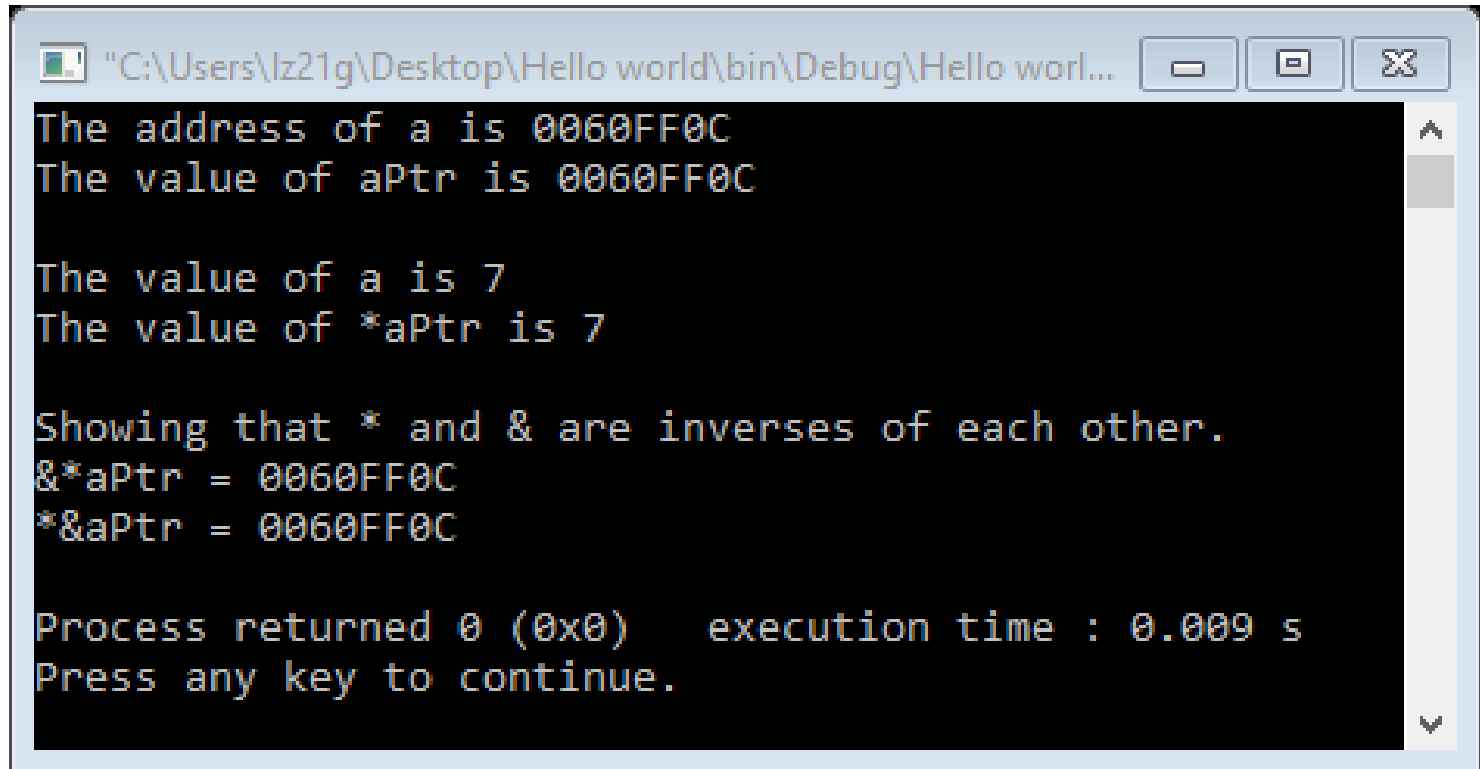
    printf( "The address of a is %p"
           "\nThe value of aPtr is %p", &a, aPtr );

    printf( "\n\nThe value of a is %d"
           "\nThe value of *aPtr is %d", a, *aPtr );

    printf( "\n\nShowing that * and & are inverses of "
           "each other.\n*&aPtr = %p"
           "\n*&aPtr = %p\n", &*aPtr, *&aPtr );

    return 0;
}
```


Example

A screenshot of a Windows command prompt window. The title bar shows the file path "C:\Users\lz21g\Desktop\Hello world\bin\Debug\Hello worl...". The window has standard minimize, maximize, and close buttons. The command prompt displays the following text:

```
The address of a is 0060FF0C
The value of aPtr is 0060FF0C

The value of a is 7
The value of *aPtr is 7

Showing that * and & are inverses of each other.
&*aPtr = 0060FF0C
*&aPtr = 0060FF0C

Process returned 0 (0x0)   execution time : 0.009 s
Press any key to continue.
```

- Address of a and the value of aPtr are identical.
- The & and * operators are complements of one another, when they are applied consecutively in either order, same result printed.

Searching a string for a character

- The `strchr()` function searches a given string for a specified character
 - First argument is the string to be searched
 - Second argument is the character you are looking for
- The function will search the string starting at the beginning and return a pointer to the first position in the string where character is found.
 - The address of this position in memory
- To store the value that's returned, you must create a variable that can store the address of a character (i.e., pointer of type `char`)
- If the character is not found, the function returns a special value `NULL`.

Example of using strchr()

```
char str[] = "The quick brown fox"; // string to be searched
```

```
char ch = 'q'; // The character we are looking for
```

```
char *pGot_char = NULL; // Pointer initialized to NULL
```

```
pGot_char = strchr(str, ch); // stores address where ch is found
```

pGot_char will point to the value ("quick brown fox")

Searching for a substring

- `strstr()` function searches for the first occurrence of the substring specified in target string.
 - If no Match it returns `NULL`

`char text[] = "Every dog has his day";` // string to be searched

`char word = "dog";` // The substring we are looking for

`char *pFound = NULL;` // Pointer initialized to `NULL`

`pFound = strstr(text, word);` // stores address where word is found

`pFound` will point to the value ("dog has his day")

Tokenization

- `strtok()` function is used for tokenizing a string
 - Requires two arguments : string to be tokenized and the string containing all the possible delimiter characters

Remember:

The first call to `strtok` must pass the C string to tokenize, and subsequent calls must specify `NULL` as the first argument, which tells the function to continue tokenizing the string you passed in first..

```
3  #include <stdio.h>
4  #include <string.h>
5
6  int main()
7  {
8      /* initialize array string */
9      char string[] = "This is a sentence with 7 tokens";
10     char *tokenPtr; /* create char pointer */
11
12     printf( "%s\n%s\n\n%s\n",
13            "The string to be tokenized is:", string,
14            "The tokens are:" );
15
16     tokenPtr = strtok( string, " " ); /* begin tokenizing sentence */
17
18     /* continue tokenizing sentence until tokenPtr becomes NULL
19     while ( tokenPtr != NULL ) {
20         printf( "%s\n", tokenPtr );
21         tokenPtr = strtok( NULL, " " ); /* get next token */
22     } /* end while */
23
24     return 0; /* indicates successful termination */
25
26 } /* end main */
27
```

The string to be tokenized is:
This is a sentence with 7 tokens

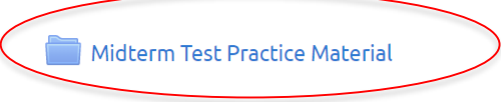
The tokens are:
This
is
a
sentence
with
7
tokens



[Clickable Link to Survey](#)

Format of Mid-term Exam

- Mid-term exam will be held on 17th November 2019 (Sunday) between 14:00 to 15:00.
- The mid term exam would consist of multiple choice questions. Go to moodle to find the sample material.




Midterm Test Practice Material


Dear Students,

Attached you will find sample questions along with their solutions in a separate file. Please note that the difficulty level of the questions in the midterm will be similar to these questions. **HOWEVER, the midterm WILL ONLY CONTAIN MULTIPLE CHOICE QUESTIONS**

We recommend you all to go through these questions before coming to the exam day. You can also post your concerns on the Programming Help section of this Moodle page.

Good Luck!

 [Announcements](#)

 [Programming Help Forum](#)

You are encouraged to post their programming questions and concerns to this forum. You are also encouraged to post replies if you feel you can help your peers.