**C String Operators:  
Writer:** Sayak Haldar

**1) The return-type used in String operations are.**

a)void only

b)void and (char \*) only

c)void and int only

d)void, int and (char \*) only

**Answer is d)** void, int and (char \*) only

**2)String operation such as strcat(s, t), strcmp(s, t), strcpy(s, t) and strlen(s) heavily rely upon.**

a)Presence of NULL character

b)Presence of new-line character

c)Presence of any escape sequence

d)None of the mentioned

**Answer is a)** Presence of NULL character

**3)Which pre-defined function returns a pointer to the last occurrence of a character in a string?**

a)strchr(s, c);

b)strrchr(s, c);

c)strlchr(s, c);

d)strfchr(s, c);

**Answer is b)strrchr(s, c)**

**Explanation:**

char \*strchr(const char \*s, int c)

this returns a pointer to the first occurrence of a string

char \*strrchr(const char \*s, int c)

this returns a pointer to the last occurrence of a string

**One example of using strchr:  
  
3)Answer is b)strrchr(s, c)**

Explanation:

char \*strchr(const char \*s, int c)

this returns a pointer to the first occurrence of a string char \*strrchr(const char \*s, int c)

this returns a pointer to the last occurrence of a string

**4) Which of the following function compares 2 strings with case-insensitively?**

a)strcmp(s, t)

b)strcmpcase(s, t)

c)strcasecmp(s, t)

d)strchr(s, t)

**Answer is c) strcasecmp(s,t)**

**5)What will be the value of var for the following? var = strcmp(“Hello”, “World”);**

a)-1

b)0

c)1

d)strcmp has void return-type

**Answer is a) -1**

Explanation: Hello is lexicographically smaller than World. Since, 'H'<'W'

**6)What is the output of this C code?**

1.#include <stdio.h>

2.int main()

3.{

4. char str[10] = "hello";

5. char \*p = strrchr(str, 'l');

6. printf("%c\n", \*(++p));

7. return 0;

8.}

a)l

b)o

c)e

d)Compilation error

**Answer is b)** o

Explanation: p points to a string “lo”. Now, we increase pointer p by 1.

\*(p+1)=p[1]

which is o

**7.What is the output of this C code?**

1.#include <stdio.h>

2.int main()

3.{

4. char \*str = "hello, world";

5. char \*str1 = "hello, world";

6. if (strcmp(str, str1))

7. printf("equal");

8. else

9. printf("unequal");

10. return 0;

11.}

a)equal

b)unequal

c)Compilation error

d)Depends on the compiler

7)Answer is b) unequal

**8.What is the output of this C code?**

1.#include <stdio.h>

2.int main()

3.{

4. char \*str = "hello";

5. char str1[5];

6. strcpy(str, str1);

7. printf("%s", str1);

8.}

a)Compilation error

b)Undefined behaviour

c)hello, world

d)Segmentation fault.

8)Answer is d) segmentation fault

Explanation: Since, str (character pointer) points to the starting location of a read only memory array (which contains the string “hello” ). That is why, in case of a character pointer, a string could be written to the memory location pointed by the character pointer only at the time of declaration of str. (i.e. Initialization) We cannot write anything in that location later. Writing a str in the location later causes segmentation fault.

**9.What is the output of this C code?**

1.#include <stdio.h>

2.#include <string.h>

3.int main()

4.{

5. char \*str = "hello, world";

6. char str1[9];

7. strncpy(str1, str, 9);

8. printf("%s %d", str1, strlen(str1));

9. return 0;

10.}

a)hello, world 11

b)hello, wor 9

c)Undefined behaviour

d)Compilation error

**Answer is c) Undefined Behaviour Explanation:**

char \*strncpy(char \*dest, const char \*src, size\_t n);

The strncpy() function is similar, except that at most n bytes of src are copied. Warning: If there is no null byte among the first n bytes of src, the string placed in dest will not be null-terminated.

If the length of src is less than n, strncpy() writes additional null bytes to dest to ensure that a total of n bytes are written.

Now, here since str1 would clearly not be null terminated, printing it with %s would invoke undefined behaviour

**10.What is the output of this C code?**

1. #include <stdio.h>

2.int main()

3.{

4. char \*str = "hello, world\n";

5. printf("%d", strlen(str));

6. return 0;

7.}

a)Compilation error

b)Undefined behaviour

c)13

d)11

**Answer is c) 13**

**11.What is the output of this C code?**

1.#include <stdio.h>

2.int main()

3.{

4. char str[11] = "hello";

5. char \*str1 = "world";

6. strcat(str, str1);

7. printf("%s %d", str, str[10]);

8. return 0;

9.}

a)helloworld 0

b)helloworld anyvalue

c)worldhello 0

d)Segmentation fault/code crash

Answer) 11)helloworld 0

**Explanation:**

char \*strcat(char \*dest, const char \*src);

So, str would act as destination string.h

Now, clearly helloworld 0 would be printed. (why 0 as str[10]? since, str[10]='\0')

However, As I suspect, a warning would be thrown for line 5 saying that incompatible implicit declaration of built-in function ‘strcat’ [enabled by default] Since, we do not include the string.h header file

**12. Strcat function adds null character**

a)Only if there is space

b)Always

c)Depends on the standard

d)Depends on the compiler

**Answer is b) Always**

**13.What is the output of this C code?**

1.#include <stdio.h>

2.int main()

3.{

4. char str[10] = "hello";

5. char \*str1 = "world";

6. strncat(str, str1, 9);

7. printf("%s", str);

8. return 0;

9.}

a)helloworld

b)Undefined behaviour

c)helloworl

d)hellowor

**Answer) a) helloworld**

char \*strncat(char \*dest, const char \*src, size\_t n); The strncat() function is similar as strcat, except that

\*it will use at most n bytes from src; and

\*src does not need to be null-terminated if it contains n or more bytes.

As with strcat(), the resulting string in dest is always null-terminated.

Now, as we can see, in case of strncat(str, str1, 9), str acts as destination string, str1 acts as source string

and 9 means at most 9 characters of source string can atmost be concatenated.

So, the whole string would be comfortably concatenated in the destination string.

However, there are two genuine queries.

First, str is of 10 characters. So, how does it fit?

I mean how does it make the final dest string as null terminated?

**I tried to print the following:**

**#include <stdio.h>**

**int main()**

**{**

**char str[10] = "hello";**

**char \*str1 = "world";**

**strncat(str, str1, 9);**

**int i;**

**for(i=0;i<11;i++)**

**{**

**printf("position %d character %c\n",i,str[i]);**

**}**

**return 0;**

**}**

Now, this is printing:

position 0 character h

position 1 character e

position 2 character l

position 3 character l

position 4 character o

position 5 character w

position 6 character o

position 7 character r

position 8 character l

position 9 character d

position 10 character

**So, where is the concept of null termination?**

Now, I just realised, the following two things:

1. **#include<stdio.h>**
2. **#include<string.h>**
3. **int main()**
4. **{**
5. **char str[5]={'H','e','l','l','o'};**
6. **printf("The str is %s\n",str);**
7. **return 0;**
8. **}**

This prints the following:

The str is Hell. Actually it is undefined behaviour, right? (that is what I know when we try to print a character array using %s when it is not null terminated)