**Sizeof operator  
Writer:** Sayak Haldar

**1. What is the sizeof(char) in a 32-bit C compiler?**

a. 1 bit

b. 2 bits

c. 1 Byte

d. 2 Bytes

**Answer is c)** 1 byte

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

**1.#include <stdio.h>**

**2.printf("%d", sizeof('a'));**

a.1

b.2

c.4

d.None of the mentioned

**Answer is c)** 4

Explanation: sizeof('a'): Here, sizeof operator will treat 'a' as 97 (ASCII value corresponding to the character. This thing happens if we are trying to find the size of a character constant

**3. Size of an array can be evaluated by:**

(Assuming array declaration int a[10];)

a.sizeof(a);

b.sizeof(\*a);

c.sizeof(a[10]);

d.10 \* sizeof(a);

**Answer is a)** sizeof(a)

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

1.#include <stdio.h>

2.union temp

3.{

4. char a;

5. char b;

6. int c;

7.}t;

8.int main()

9.{

10. printf("%d", sizeof(t));

11. return 0;

12.}

a.1

b.2

c.4

d.6

**Answer is c)** 4

**Explanation:** Size of an identifier defined within the union name space is always =maximum size(member of union). Here, the maximum size of a member of union=size of int member/size of float member

**5. Which among the following has the highest precedence?**

a.&

b.<<

c.sizeof()

d.&&

**Answer is c)** sizeof()

**6. Which of the following is not an operator in C?**

a.,

b.sizeof()

c.~

d.None of the mentioned

**Answer is d)**Comma is an operator.  
sizeof() is an operator.

~ is bitwise negation operator.

**7. The sizeof(void) in a 32-bit C is\_\_\_\_\_.**

a.0

b.1

c.2

d.4

**Answer is b)** 1

**8. What type of value does sizeof return?**

a.char

b.short

c.unsigned int

d.Long

**Answer is c) unsigned int**

**9.What will be the size of the following structure?**

1.#include <stdio.h>

2.struct temp

3.{

4. int a[10];

5. char p;

6.};

a.5

b.11

c.41

d.44

**Answer is c) 41**

**10.Comment on the output of following C program?**

1.#include <stdio.h>

2.main()

3.{

4. int a = 1;

5. printf("size of a is %d, ", sizeof(++a));

6. printf("value of a is %d", a);

7. return 0;

8.};

a. size of a is 4, value of a is 1

b. size of a is 4, value of a is 2

c. size of a is 2, value of a is 2

d. size of a is 2, value of a is 2

**Answer) is a)**

Explanation: The sizeof operator yields the size (in bytes) of its operand, which may be an expression or the parenthesized name of a type. The size is determined from the type of the operand. The result is an integer. If the type of the operand is a variable length array type, the operand is evaluated; otherwise, the operand is not evaluated and the result is an integer constant.

Or, in a simple language.

sizeof is a compile-time operator, so at the time of compilation sizeof and its operand get replaced by the result value. The operand is not evaluated (except when it is a variable length array) at all; only the type of the result matters.

**11. Which among the following is right?**

a. sizeof(struct stemp\*) > sizeof(union utemp\*) > sizeof(char \*) b. sizeof(struct stemp\*) < sizeof(union utemp\*) < sizeof(char \*) c. sizeof(struct stemp\*) = sizeof(union utemp\*) = sizeof(char \*) d. The order Depends on the compiler

**Answer is c)** sizeof(struct stemp\*) = sizeof(union utemp\*) = sizeof(char \*)

**Explanation:** Since, pointer size is not depended upon datatype of the pointer, it is solely dependedupon compiler (32 bit compiler or 64 bit compiler)

**12.Comment on the following C code?**

1.#include <stdio.h>

2.printf("%d", sizeof(strlen("HELLOWORLD")));

a. Output, 4

b. Output, 10

c. Output, 16

d. Error, sizeof cannot evaluate size of a function.

**Answer) is a)**however, remember, sizeof() is a compile time operator

**13. Which of the following cannot be used inside sizeof?**

a.pointers

b.functions

c.macro definition

d.None of the mentioned

**13) Answer is d) None of the mentioned**

Explanation: In case of pointers, sizeof(pointer) will return 4 if it is a 32 bit compiler and sizeof(pointer) will return 8 if it is a 64 bit compiler

In case of functions, sizeof(function)=sizeof(return type of the function)

In case of macros, sizeof(macro) =sizeof(the datatype of the constant value associated with macro)

**14.Comment on the following C code?**

1.#include <stdio.h>

2.//(sizeof double = 8, float = 4, void = 1)

3.#define PI 3.14

4.int main()

5.{

6. printf("%d", sizeof(PI));

1. return 0;

8.}

a. Output is 8

b. Output is 4

c. Output is 1

d. Error, we can’t use sizeof on macro-definitions

**Answer)a) Output is 8)**

**Explanation:** Since, PI will simply be replace with the value 3.14 after preprocessing and 3.14 is a double constant

1. **Comment On The Following C Code:**
2. **#include<stdio.h>**
3. **int main()**
4. **{**
5. **char \*str="Sayak";**
6. **printf("size of the string pointed by str pointer %d\n",(int)sizeof(str));**
7. **return 0;**
8. **}**

a)4

b)8

c)5  
d) invalid operand for sizeof

**Answer) is a or b depended on the 32 bit or 64 bit size of the compiler.  
  
16. Comment On the Following Code:**

1. **#include<stdio.h>**
2. **int main()**
3. **{**
4. **char str[]="Sayak";**
5. **printf("size of the str character array %d\n",(int)sizeof(str));**
6. **//printf("%s",str);**
7. **return 0;**
8. **}**

a)5  
b)6  
c)4  
d)8  
  
**Answer)**

Now, first you have to understand, what does this initialization mean?

“Sayak” is stored in an array from unnamed read only memory location in the following way:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| S | a | y | a | k | \0 |

Now, str is initialized to that unnamed character array. Now, str is not a read only array. We can write on it. Now, str is an array of 6 characters. So, sizeof(str) will return 6. However, strlen in this case will return 5, because of the internal implementation of strlen.

**Hence, result is b) 6.**

**17.Comment On The Output Of The Following Code:**

1. #include<stdio.h>
2. int main()
3. {
4. char str[]={'S','a','y','a','k'};
5. printf("size of the str character array %d\n",(int)sizeof(str));
6. return 0;
7. }

a)5  
b)6  
c)4  
d)8

**Answer)** the str character array has only 5 characters. Hence, 5.

**Note: The sizeof operator does not produce undefined behaviour (like, strlen, strcpy etc) if the char array is not null terminated.**