DATA STRUCTURE & ALGORITHM

Solutions

1. The minimum number of temporary variables needed to swap the contents of two variables is:

(a) 1 (b) 2
(c) 3 (d) 0

Solution: Option (d)

Explanation:

Suppose there are two variables a and b whose values needs to be swapped. Follwing the below procedure we can swap the values in two variables.

a=a+b

b=a - b

a=a - b

2. Consider the function:
find (int x, int y)
{ return((x < y) ? 0: (x – y)); }

Let a, b be two non-negative integers. The call find(a, find(a, b)) can be used to find the:
(a) maximum of a, b (b) positive difference of a, b
(c) sum of a, b (d) minimum of a, b

Solution: Option (d)

3. The following:
printf (“%f”, 9/5);

prints:
(a) 1.8 (b) 1.0
(c) 2.0 (d) none of the above

Solution: Option (d)

Explanations:
9/5 yeilds integer 1. Printing 1 as a floating point number prints garbage.

4. If an integer needs two bytes of storage then maximum value of unsigned integer is:

(a) $2^{16}–1$ (b) $2^{15}–1$
(c) 2^{16} (d) 2^{15}

Solution: Option (a)

5. If an integer needs two bytes of storage then maximum value of a signed integer is:

(a) $2^{16}–1$ (b) $2^{15}–1$
(c) 2^{16} (d) 2^{15}

Solution: Option (b)

Explanation:
In signed magnitude form, one bit is dedicated to store the sign. (e.g., 1 for negative and 0, otherwise). Only the remaining 15 bits are available to store the magnitude. Hence the answer.

6. printf(“%d”, printf(“tim”));

(a) results in a syntax error (b) outputs tim3
(c) outputs garbage (d) prints tim and terminates abruptly

Solution: Option (b)

Explanation:
Any function (including main( )), returns a value to the calling environment. In case of the printf, it is the characters it printed. So, the output will be tim3 (since it printed the three characters a, b, c).

7. If a b c is the input then the following program fragment results in:
char x, y, z;

printf(“%d”, scanf(“%c %c %c”, &x, &y, &z));

results in:
(a) a syntax error
(b) a fatal error
(c) segmentation violation
(d) printing of 3

Solution: Option (d)

Explanation:
The scanf function returns the number of successful matches i.e., 3 in this case.

8. Consider the statements:
putchar(getchar( ));
putchar(getchar( ));

if a
b
is the input, the output will be:
(a) an error message
(b) this can’t be the input
(c) ab
(d) a b

Solution: Option (b)

9. Let a, b be two positive integers, which of the following options correctly relates / and %?

(a) b= (a/b) * b + a%b
(b) b= (a%b) * b + a/b
(c) a= (a/b) * b + a%b
(d) a= (a%b) * b + a/b

Solution: Option (c)

Explanation:

Lets consider two cases when a is a mutiple of b and a is not a multiple of b. (a/b) results in integer division of a by b. In case a is completely divisible by b then a/b gives the quotient on dividing a by b and multiplying that with b again gives a, If a is not evenly divisible by b then (a/b)*b will give the nearest multiple of b which is less than a. To that we add (a%b) which is the remainder when a is divided by b , to finally get a.

10. Consider the following program fragment:
char c= ‘a’
`while (c++ ≤ ‘z’)`
putchar (xxx);

if the required output is abcdefghijklmnopqrstuvwxyz then xxx should be:
(a) c (b) c++
(c) c–1 (d) –c

Solution: Option (c)

11. If y is of integer type then the expressions:
3 * (y–8)/9 and (y–8)/9 * 3

(a) must yield same value (b) must yield different values
(c) may or may not yield the same value (d) none of the above

Solution: Option (c)

Explanation:

If (y-8) is evenly divisible by 9 then the result of both expression will be same.

12. The statement:
```
if (my Ptr != NULL)
   *myPtr= NULL;
else
   *myPtr= NULL;
```

has the same effect as the statement(s):
(a) if (myPtr) * myPtr= NULL; (b) *myPtr= NULL;
else *myPtr= NULL;
(c) if (!myPtr) *myPtr= NULL; (d) All of the above
else *myPtr= NULL;

Solution: Option (d)

13. The following code fragment:
int x, y= 2, z, a;

```
x= (y* =2) + (z= a =y);
printf("%d", x);
```

(a) prints 8
(b) prints 6
(c) prints 6 or 8 depending on the compiler implementation
(d) is syntactically wrong

Solution: Option (c)

Explanation:

Given statement is x= (y* =2) + (z= a =y);

If the compiler uses left associativity then first y will be multiplied by 2 hence value of y will be made 4. After that the expression z=a=y , will make value of z 4. Hence x is 4+4 =8. If the compiler uses right associativity then first z will be made 2, after that y will be doubled . Therefore we get x=4+2 =6. So the result depends on the implementation.

14. If n has the value 3 then the output of statement:

```
printf("%d %d", n++, ++n);
```

(a) is 3 5
(b) is 4 5
(c) is 4 4
(d) is implementation dependent

Solution: Option (d)

15. x– = y+1; does the same as:

(a) x= x –y +1
(b) x= – x –y – 1
(c) x= –x + y +1
(d) x= x – y – 1

Solution: Option (d)

16. The expression 5 – 2 – 3 * 5 – 2 will evaluate to 18, if:

(a) – is left associative and * has precedence over –
(b) – is right associative and * has precedence over –
(c) – is right associative and – has precedence over *

(d) – is left associative and – has precedence over *

Solution: Option (c)

Explanation:
5 – 2 – 3 * 5 – 2 will yield 18, if it is treated as (5 – (2 – 3)) * (5 – 2) i.e. if – has precedence over * and if it associates from the right.

17. printf ("%c", 100);

(a) prints 100
(b) prints the ASCII equivalent of 100
(c) prints garbage
(d) none of the above

Solution: Option (b)

18. The program fragment:

```
int i= 263;
putchar (i)
```

(a) prints 263
(b) prints the ASCII equivalent of 263
(c) rings the bell
(d) prints garbage

Solution: Option (c)

Explanation:
263 in binary form is 100000111. If one tries to print an integer as a character, only the last 8 bits will be considered, the rest chopped off. So, in this case the ASCII value of 0000111 (i.e., decimal 7) will be printed. Look in the ASCII table. It is ringing the bell!