第47回

C言語プログラミング能力認定試験

３　級

正答・解説

サーティファイ
情報処理能力認定委員会

第47回正答　解説

＜解説＞

問１

(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
イ	イ	イ	ア	ア	ア	イ	ア

(1) C言語は，特定の計算機のアーキテクチャとは独立しているので，異なるハードウェアで動作させることのできる「移植可能なプログラム」を書くことができる。

(2) C言語では注釈文を記述することができ，/*と*/の間に書く。

(3) C言語では，#include 前処理命令によって読み込む対象のソースファイル内に，さらに#include 前処理命令を記述することができる。

(7) switch 文の「case 定数式」で指定した定数式のいずれにも一致しないときの処理は，default ラベルに記述することができる。

問２

(9)	(10)	(11)	(12)	(13)
エ	オ	イ	ア	エ

(9) 配列 str1 には，初期化式として"PARITYBIT"が指定されている。文字列"PARITY BIT"による初期化は，{'P', 'A', 'R', 'I', 'T', 'Y', 'B', 'I', 'T', '¥0'} という初期化リストを指定したものと等価である。したがって，配列 str1 の要素数は 10 となる。

(10) 配列 str1 の各要素の値は，次のようになる。したがって，str1[5]は'Y'となる。

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
str1	'P'	'A'	'R'	'I'	'T'	'Y'	'B'	'I'	'T'	'¥0'

(11) 配列 str2 には，初期化式として {'C', 'H', 'E', 'C', 'K', 'S', 'U', 'M'} が指定されている。したがって，その要素数は 8 となる。

(12) 配列 pb には，初期化式として {{4, 3, 1, 5, 9}, {8, 4, 5, 2, 1, 7, 3}, {1, 2, 4, 7, 3, 6}} が指定されている。したがって，3 × 8 の配列となる。

(13) 配列 pb の各要素の値は，次のようになる。ここで，初期化リストで不足している領域は 0 で初期化される。

配列pb

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]
[0]	4	3	1	5	9	0	0	0
[1]	8	4	5	2	1	7	3	0
[2]	1	2	4	7	3	6	0	0

したがって，pb[1][2]は 5 となる。

問３

(14)	(15)	(16)	(17)	(18)	(19)	(20)	(21)
イ	ウ	エ	イ	ア	イ	ウ	エ

(14)，(15) 変数 x，y，v の値をトレースすると，次のようになる。

文	x	y	v
x = 5; y = 8; v = 0;	5	8	0
while (x < y)　　･･･「真」	5	8	0
v += 1;	5	8	1
x += v;	6	8	1
while (x < y)　　･･･「真」	6	8	1
v += 1;	6	8	2
x += v;	8	8	2
while (x < y)　　･･･「偽」	8	8	2

(16)，(17) 変数 x，y，v の値をトレースすると，次のようになる。

文	x	y	v
x = 5; y = 2; v = 3;	5	2	3
while (x >= y)　　･･･「真」	5	2	3
v += 1;	5	2	4
x -= v;	1	2	4
while (x >= y)　　･･･「偽」	1	2	4

(18)，(19) 変数 x，y，v の値をトレースすると，次のようになる。

文	x	y	v
x = 1; y = 2; v = 3;	1	2	3
do	1	2	3
v += 1;	1	2	4
x -= v;	-3	2	4
while (x > y);　　･･･「偽」	-3	2	4

(20)，(21) 整数配列 data の内容は，次のようになっている。

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]
data	3	9	1	5	2	8	4	6	7

変数 x，y，v の値をトレースすると，次のようになる。

文	x	y	v	x - y	data[x - y]
x = 7; y = 0; v = -3;	7	0	-3		
while (x > y)　・・・「真」	7	0	-3		
v -= data[x - y];	7	0	-9	7	6
do	7	0	-9		
x -= 1;	6	0	-9		
v += 8;	6	0	-1		
while (v < 0); ・・・「真」	6	0	-1		
do	6	0	-1		
x -= 1;	5	0	-1		
v += 8;	5	0	7		
while (v < 0); ・・・「偽」	5	0	7		
y += 3;	5	3	7		
while (x > y)　・・・「真」	5	3	7		
v -= data[x - y];	5	3	6	2	1
do	5	3	6		
x -= 1;	4	3	6		
v += 8;	4	3	14		
while (v < 0); ・・・「偽」	4	3	14		
y += 3;	4	6	14		
while (x > y)　・・・「偽」	4	6	14		

問 4

(22)	(23)	(24)	(25)	(26)	(27)
イ	ア	イ	エ	ウ	イ

前置インクリメント演算子の結果は，<u>オペランドの値を増分した後の値</u>(空欄 22)となるのに対し，後置インクリメント演算子の結果は，<u>オペランドの値を増分する前の値</u>(空欄 23)となる。

(24)「a = i++」では，変数 a に変数 i の値である 10 が格納された後，変数 i の値が＋1 されて 11 となる。したがって，「a : 10」と表示される。

(25)「b = ++i」では，変数 i の値が＋1 されて 12 になった後，変数 b に変数 i の値である 12 が格納される。したがって，「b : 12」と表示される。

(26)，(27)「c = --i」では，変数 i の値が－1 されて 11 になった後，変数 c に変数 i の値である 11 が格納される。変数 c の値も変数 i の値も 11 なので「if (c > i)」は「偽」となり，「d = i--」が実行される。「d = i--」では，変数 d に変数 i の値である 11 が格納された後，変数 i の値が－1 されて 10 となる。したがって，「d : 11」，「i : 10」と表示される。

問５

(28)	(29)	(30)	(31)	(32)
エ	エ	ア	エ	エ

(28) "%-4d"という書式を指定すると，10 進数の左寄せ 4 桁で出力され，余った桁には空白が出力される。

(29) "%04x"という書式を指定すると，16 進数の右寄せ 4 桁で出力され，余った桁には'0'が出力される。ここで，"x"のように英小文字で書式を指定すると，16 進数の'A'～'F'の値は英小文字で出力される。

(30) "%8s"という書式を指定すると，8 文字分の領域に右寄せで出力され，余った桁には空白が出力される。

(31) "%-3s"という書式を指定すると，3 文字分の領域に左寄せで出力され，余った桁には空白が出力される。ここで，指定した文字列が 3 文字より長い場合は，文字列の末尾まで出力される。

(32) "%-7.5s"という書式を指定すると，文字列の先頭 5 文字が 7 文字分の領域に左寄せで出力され，余った桁には空白が出力される。

問６

(33)	(34)	(35)	(36)	(37)
エ	イ	ア	ア	イ

(33)「while (inbuf[i] != '¥0') 」より，入力文字列 inbuf を先頭から走査するための位置（配列の要素番号）として変数 i が使用されていることが分かる。空欄（33）に入る条件が「真」の場合，キー文字列の数字を数値に変換し，その数値に対応した置換文字列を出力文字列とする処理（＜処理手順＞(3)）を行っているから，「%」の次の文字が数字（0～9）であれば「真」となる条件を入れる。したがって，「isdigit(inbuf[i + 1])」となる。

(34) キー文字列の数字部分は，「n = inbuf[i + 1] - '0'」で数値化されて，変数 n に格納される。その変数 n を添字として，for 文内の「outbuf[j] = rep[n][k]」により，置換文字列を出力文字列へ転記する。rep[n]の先頭から末尾まで転記するための for 文の初期化式は k = 0, 再設定式は k++であるから，継続条件は，「rep[n][k]が'¥0'でない」となる。したがって，「rep[n][k] != '¥0'」となる。

(35) ＜処理手順＞(4)の「走査対象文字が「%」で，かつ，次の文字も「%」であれば，「%」１文字を出力文字列とする」処理である。したがって，「inbuf[i + 1] == '%'」となる。

(36) 置換が正常に終了した場合（eflg == 0），while 文の反復処理が終了した時点では，出力文字列（outbuf）の末尾に'¥0'が格納されていない。しかし，次行で printf 関数により文字列として出力するためには，出力文字列（outbuf）の末尾に'¥0'が格納されていなければならない。出力文字列（outbuf）の末尾は変数 j が示している。したがって，「outbuf[j] = '¥0'」となる。

(37) 不正な文字を検知した場合（eflg == 1)に,「%」の次の文字を不正文字として出力する処理（＜処理手順＞(5)）である。入力文字列（inbuf）の中から不正文字が検知されて while 文の反復処理が終了した時点で，その不正文字の位置は i + 1 となっている。したがって，「inbuf[i + 1]」となる。