Lista de Revisão - PIF - AV 1 - Parte 2

Instruções

- Tipos: (ME) múltipla escolha, (CS) análise de código/saída, (VF) verdadeiro ou falso (assinale V ou F e justifique quando F), ou implementação de código.
- **Linguagem de referência: C.**

1. (ME) Qual caractere marca o fim de uma string em C?
   a) '\n'
   b) '\0'
   c) '\r'
   d) 'EOF'
2. (ME) Qual biblioteca padrão contém as funções `strlen`, `strcpy` e `strcat`?
   a) stdio.h
   b) stdlib.h
   c) string.h
   d) ctype.h
3. (VF) O vetor `char nome[5] = "João";` está corretamente dimensionado para armazenar a string e o caractere nulo.
4. (ME) Dado `char s[] = "CESAR";`, qual é o valor retornado por `strlen(s)`?
   a) 4
   b) 5
   c) 6
   d) Indefinido
5. (CS) Qual a saída?

```
char txt[20] = "Oi";
strcat(txt, "!");
printf("%s", txt);
```

6. (VF) A função `gets()` é recomendada por ser segura contra estouro de buffer.
7. (ME) Para ler uma linha inteira (até a quebra de linha) usando `scanf`, o formato correto é:
   a) "%s"
   b) "%[A-Z]"
   c) "%[^]"
   d) "%c"
8. (CS) Saída do código:

```
char a[] = "abc";
char b[] = "abc";
int r = strcmp(a, b);
printf("%d", r);
```

9. (VF) `tolower('A')` retorna 'a'.

10. (ME) Qual função copia no máximo N caracteres de uma string origem para o destino?
a) strcpy
b) strncpy
c) memcpy
d) strncat

11. (CS) Saída:

```
char s[10] = "abc";
printf("%lu", sizeof(s));
```

12. (VF) O código ASCII do caractere '0' é menor que o código de 'A'.

13. (ME) Função que converte todos os caracteres de uma string para maiúsculo:
a) strupr
b) strlwr
c) toupper
d) isupper

14. (CS) Saída:

```
char x = 'a';
if(isdigit(x)) printf("digito\n");
else printf("nao\n");
```

15. (VF) `isspace('\t')` devolve verdadeiro.

16. (ME) Qual destas não é função da `ctype.h`?
a) isalpha
b) islower
c) strcasecmp
d) ispunct

17. (CS) Saída:

```
char p[10] = "C";
strncat(p, "ESAR", 2);
printf("%s\n", p);
```

18. (VF) `strcmp("ABC", "abc")` devolve 0 em sistemas sem distinção de maiúsculas/minúsculas.

19. (ME) A expressão `'a' - 'A'` vale:
a) 0
b) 26
c) 32
d) 65

20. (CS) Saída:

```
char s[] = "12345";
printf("%c", s[ strlen(s) - 1 ]);
```

21. (VF) Uma string declarada como constante (entre aspas) já inclui automaticamente o '\0'.

22. (ME) Para copiar exatamente 5 caracteres, incluindo o nulo, de `orig` para `dest`, usa-se:

a) `strncpy(dest, orig, 5);`

b) `strcpy(dest, orig, 5);`

c) `memcpy(dest, orig, 5);`

d) `strncat(dest, orig, 5);`

23. (CS) Saída:

```
char c = '\n';
printf("%d", isspace(c));
```

24. (VF) A função `puts` adiciona automaticamente uma quebra de linha após imprimir a string.

25. (ME) Qual código imprime o tamanho de "PIF" seguido de uma quebra de linha?

a) `printf("%d\n", strlen("PIF"));`

b) `puts(strlen("PIF"));`

c) `printf("%s", strlen("PIF"));`

d) `printf("%lu", sizeof("PIF"));`

26. (CS) Saída:

```
char s[] = "abc";
printf("%c", *(s+1));
```

27. (VF) `sizeof("abc")` devolve 3.

28. (ME) Em `char nome[100];`, quantos bytes são reservados?

a) 99

b) 100

c) 101

d) Depende do SO

29. (CS) Saída:

```
char t[4] = {'c','o','d','e'};
printf("%s", t);
```

30. (VF) `strcat` não verifica se há espaço suficiente no destino.

31. (ME) Qual função localiza a primeira ocorrência de um caractere em uma string?

a) strchr

b) strstr

c) strtok

d) strpbrk

32. (CS) Saída:

```
char *p = strchr("banana", 'n');
printf("%s", p);
```

33. (VF) `strtok` modifica a string original ao tokenizar.

34. (ME) A tabela ASCII define quantos caracteres básicos?

a) 64
b) 95
c) 128
d) 256

35. (CS) Saída:

```
printf("%c", "ABC"[0]);
```

36. (ME) Qual palavra-chave indica que uma função não retorna valor?

a) null
b) void
c) none
d) empty

37. (VF) Funções em C devem ser declaradas dentro da função `main`.

38. (CS) Saída:

```
int soma(int a,int b){return a+b;}
int main(){printf("%d", soma(3,4));}
```

39. (ME) O tipo de `sqrt` definido em `math.h` é:

a) void
b) int
c) double
d) float

40. (VF) Variáveis declaradas fora de qualquer função têm escopo global.

41. (ME) Qual é a forma correta de prototipar uma função que recebe um `char*` e devolve `int`?

a) `int f(char*);`
b) `f int(char*);`
c) `void f(char*);`
d) `char* f(int);`

42. (CS) Saída:

```
void inc(int *p){(*p)++;}
int main(){int x=5;inc(&x);printf("%d",x);}
```

43. (VF) É possível declarar duas funções com o mesmo nome e parâmetros em C (overload).

44. (ME) Em C, argumentos são passados:

a) Sempre por referência
b) Sempre por valor
c) Por valor, mas ponteiros permitem efeito de referência
d) Depende do compilador

45. (CS) Saída:

```
int f(){static int c=0;return ++c;}
```

```
int main(){printf("%d %d",f(),f());}
```

46. (VF) Uma função pode ser chamada antes de sua declaração se houver protótipo.

47. (ME) Qual modificador armazena uma variável local na memória estática preservando seu valor?

a) const
b) register
c) static
d) volatile

48. (CS) Saída:

```
void show(int n){printf("%d",n);}
int main(){printf("%d", show(3));}
```

49. (VF) Um procedimento é simplesmente uma função que retorna `void`.

50. (ME) Qual opção representa corretamente uma chamada de função embutida (nested call)?

a) `printf(sqrt(16));`
b) `printf("%f", sqrt(16));`
c) `sqrt(printf("16"));`
d) `void(sqrt(16));`

51. (CS) Saída:

```
int x=10;
int soma(int a){return a+x;}
int main(){int x=5;printf("%d",soma(3));}
```

52. (VF) Variáveis globais podem ser redeclaradas dentro de uma função sem erro.

53. (ME) Qual diretiva inclui a biblioteca com `scanf`?

a) `#include <stdio.h>`
b) `#include <stdlib.h>`
c) `#include <string.h>`
d) `#include <math.h>`

54. (CS) Saída:

```
void foo(){int x=1;}
int main(){foo();printf("ok");}
```

55. (VF) O `return` em `main` comunica ao sistema operacional o status de execução.

56. (ME) Assinale a afirmativa correta:

a) Toda função deve ter ao menos um parâmetro
b) `void main()` é padrão ANSI C
c) O compilador aceita múltiplos `return` na mesma função
d) Uma função não pode chamar outra

57. (CS) Saída:

```
int f(int n){if(n==0)return 1;return n*f(n-1);}
```

```
int main(){printf("%d",f(4));}
```

58. (VF) Uma função recursiva deve sempre ter um caso base.

59. (ME) Qual palavra-chave impede que uma variável global seja visível em outros arquivos?

a) static
b) extern
c) auto
d) const

60. (CS) Saída:

```
int g(int a,int b){return a>b?a:b;}
int main(){printf("%d",g(2,7));}
```

61. (VF) A instrução `scanf("%d", x);` está correta para ler um int.

62. (ME) Em `int (*pf)(int,int);`, `pf` é:

a) Ponteiro para função
b) Função que retorna ponteiro
c) Array de ponteiros
d) Ponteiro para int

63. (CS) Saída:

```
int add(int a,int b){return a+b;}
int main(){int (*p)(int,int)=add;printf("%d",p(1,2));}
```

64. (VF) O escopo de uma variável `register` é sempre global.

65. (ME) Qual é o valor de retorno padrão de `main` quando omitido?

a) 1
b) 0
c) -1
d) Indefinido

66. (ME) Qual palavra-chave define uma estrutura em C?

a) record
b) struct
c) object
d) class

67. (VF) Todos os membros de uma struct ocupam o mesmo endereço de memória.

68. (CS) Saída:

```
struct P{int x,y;}p={1,2};
printf("%d",p.y);
```

69. (ME) Para criar um alias `Pessoa` para uma struct, usa-se:

a) `alias Pessoa struct {...};`
b) `typedef struct {...} Pessoa;`
c) `struct Pessoa {...};`
d) `struct {...} Pessoa;`

70. (VF) Podemos declarar um vetor de structs.

71. (ME) Dada `Cachorro dog;`, o acesso ao campo `raca` é:
a) `dog->raca`
b) `dog.raca`
c) `dog[raca]`
d) `raca.dog`

72. (CS) Saída:

```
typedef struct{int h,m,s;}Hora;
Hora t={1,2,3};
printf("%02d:%02d",t.h,t.m);
```

73. (VF) É possível ter um membro de struct que seja outra struct.

74. (ME) Uma union difere de uma struct porque:
a) Não possui membros
b) Ocupa espaço para todos os membros
c) Compartilha a mesma área de memória
d) Só aceita tipos inteiros

75. (CS) Saída:

```
union U{int a;char c;};
union U u;u.a=65;printf("%c",u.c);
```

76. (VF) Atribuir a um membro de uma union sobrescreve os demais.

77. (ME) Qual diretiva imprime o deslocamento de bytes de um campo dentro da struct?
a) offsetof
b) sizeof
c) alignof
d) printf

78. (CS) Saída:

```
typedef union{float t;char st;}Sensor;
Sensor s; s.t=25.5; printf("%.1f",s.t);
```

79. (VF) `typedef` pode ser usado para rotular tanto structs quanto enums.

80. (ME) Enumerações em C atribuem por padrão valores inteiros começando em:
a) -1
b) 0
c) 1
d) 255

81. (CS) Saída:

```
enum dia{SEG=1,TER,QUA};
printf("%d",QUA);
```

82. (VF) É permitido ter dois membros enum com o mesmo valor numérico.

83. (ME) Para imprimir o valor 2 de um enum `cores {VERM, AZUL, VERDE};`, usa-se:
a) `printf("%s", VERDE);`
b) `printf("%d", VERDE);`

c) `printf("%c", VERDE);`
d) `printf(VERDE);`

84. (CS) Saída:

```
enum escape{TAB='\t',NL='\n'}; printf("%c",TAB);
```

85. (VF) Uma variável enum pode receber um valor inteiro fora da lista, mas é má prática.

86. (ME) Qual declaração cria um vetor de 10 structs `Aluno`?

a) `Aluno[10] alunos;`
b) `struct Aluno alunos[10];`
c) `Aluno alunos[10];`
d) `struct alunos[10];`

87. (CS) Saída:

```
struct S{int a;};
struct S s={5};
printf("%d",s.a);
```

88. (VF) Podemos inicializar parcialmente uma struct usando designadores.

89. (ME) Qual operador seleciona um membro via ponteiro para struct?

a) .
b) ->
c) &
d) *

90. (CS) Saída:

```
struct P{int x;};
struct P *pp=NULL;
printf("%p",(void*)pp);
```

91. (VF) `sizeof(struct vazio{})` é 0 em C.

92. (ME) Para criar um tipo union chamado `Numero` contendo `int i` e `float f`:

a) `union Numero{int i;float f;};`
b) `union{int i;float f;}Numero;`
c) `typedef union{int i;float f;} Numero;`
d) Ambas a e c corretas

93. (CS) Saída:

```
enum mes{JAN=1,FEV}; printf("%d",FEV);
```

94. (VF) O tamanho de uma union é igual ao tamanho de seu maior membro.

95. (ME) Qual das alternativas define corretamente uma enum para dias úteis com alias `diaUtil`?

a) `enum diaUtil{SEG,TER,QUA,QUI,SEX};`
b) `typedef enum{SEG,TER,QUA,QUI,SEX} diaUtil;`

c) `enum{SEG,TER,QUA,QUI,SEX} diaUtil;`
d) Todas acima

96. (CS) Saída:

```
struct A{char c;int i;};

printf("%zu",sizeof(struct A));
```

97. (VF) O preenchimento (padding) pode aumentar o tamanho real de uma struct além da soma de seus membros.

98. (ME) Qual expressão acessa o segundo elemento do vetor `canil` de tipo `Cachorro`?

a) `canil.1`
b) `canil[1]`
c) `canil->1`
d) `canil[2]`

99. (ME) Qual operador é usado para **obter o endereço** de uma variável?

a) *
b) &
c) ->
d) %

100. (ME) Dado `int x = 10; int *p = &x;`, o valor de `*p` é:

a) O endereço de x
b) O valor armazenado em x
c) O tipo do ponteiro
d) Indefinido

101. (VF) O operador `*` serve tanto para declarar quanto para acessar o valor apontado por um ponteiro.

102. (CS) Saída:

```
int a = 5, b = 10;
int *p = &a;
*p = *p + b;
printf("%d", a);
```

103. (ME) Se `int *p;` foi declarado mas não inicializado, então:

a) Aponta para NULL
b) Aponta para o endereço 0
c) Contém um valor indeterminado
d) Causa erro de compilação

104. (VF) Um ponteiro `void*` pode apontar para qualquer tipo de dado, mas não pode ser dereferenciado sem conversão.

105. (CS) Saída:

```
int v[3] = {2,4,6};
int *p = v;
printf("%d", *(p+1));
```

106. (ME) Qual é a forma correta de declarar um ponteiro para ponteiro para inteiro?
a) int p**;
b) int *p*;
c) int **p;
d) pointer int p;

107. (VF) p++ em um ponteiro incrementa o endereço armazenado em uma unidade de byte.

108. (ME) O que ocorre ao tentar acessar um ponteiro não inicializado?
a) Nada
b) Valor 0
c) Comportamento indefinido
d) Erro de compilação

109. (VF) A expressão &*p é equivalente a p.

110. (ME) Qual opção imprime corretamente o endereço de uma variável x?
a) printf("%d", x);
b) printf("%p", &x);
c) printf("%x", x);
d) printf("%s", &x);

111. (CS) Saída:

```
int x = 10;
int *p = &x;
printf("%p %d", p, *p);
```

112. (VF) O tipo `char *p = "texto";` cria uma string constante armazenada em área somente leitura.

113. (ME) O que acontece ao executar `free(p)` sobre um ponteiro não obtido por `malloc`?
a) Nada
b) Libera a memória
c) Gera comportamento indefinido
d) Zera o ponteiro

114. (CS) Saída:

```
int a = 4, b = 7;
int *p = &a, *q = &b;
*p = *q;
printf("%d %d", a, b);
```

115. (VF) Um array e um ponteiro para seu primeiro elemento são equivalentes em expressões aritméticas.

116. (ME) Qual expressão equivale a `v[i]` se v é um ponteiro?
a) *v + i
b) *(v + i)
c) *(v - i)
d) *v[i]

117. (CS) Saída:

```
char s[] = "PIF";
```

```
char *p = s;
printf("%c", *(p+2));
```

118. (VF) Ponteiros podem ser usados para percorrer vetores e strings sem índice.

119. (ME) Dado `int x=5, *p=&x;`, o que imprime `printf("%d", ++*p);`?

a) 5
b) 6
c) Endereço de x
d) Erro

120. (VF) Ponteiros para funções permitem armazenar endereços de funções e chamá-las dinamicamente.

121. Escreva uma função que receba um vetor de structs Aluno { char nome[40]; float notas[3]; } e devolva o índice do aluno com maior média.

122. Implemente um procedimento que converta todos os caracteres de uma string para maiúsculo sem usar o procedimento strupr.

123. Crie um programa que leia *N* números e use uma função para calcular o máximo divisor comum (MDC) de todos eles.

124. Defina uma `enum` para representar os meses do ano e escreva um programa que, dado o número do mês, imprima seu nome por extenso.

125. Escreva um código em C que leia duas horas no formato HH:MM:SS (24 h), definindo-as como uma struct e imprima a diferença entre elas em segundos.

126. Implemente uma `union` chamada `Valor` que possa armazenar `int`, `float` ou `char` e escreva um programa de demonstração que leia um tipo e um valor, armazene-o na union e imprima-o corretamente.

127. Escreva uma função que receba uma string e retorne a quantidade de dígitos, letras e outros caracteres presentes.

128. Escreva um código que determine se uma frase lida do teclado é um palíndromo, desconsiderando espaços e diferenças entre maiúsculas/minúsculas.

129. Crie um programa que leia um vetor de `struct Hora { int h, m, s; }` com horários de eventos e os ordene do mais cedo para o mais tarde.

130. Implemente uma função que imprima uma string ao contrário.

131. Escreva um procedimento `void troca(int *a, int *b)` que troque os valores das variáveis apontadas por a e b.

132. Implemente uma função `int soma_vetor(int *v, int n)` que receba um vetor de inteiros e retorne a soma dos elementos usando apenas aritmética de ponteiros.

133. Escreva uma função `char* copia_string(const char *origem)` que aloque dinamicamente memória e copie a string origem para uma nova área.

134. Crie uma função `void inverte(int *v, int n)` que inverta os elementos de um vetor de inteiros usando ponteiros.

135. Implemente uma função `float media(float *valores, int n)` que calcule a média dos elementos do vetor sem usar colchetes.

136. Implemente um procedimento `void maiusculo(char *s)` que percorra uma string e converta todas as letras para maiúsculas usando `toupper`.

137. Implemente uma função int conta_substr(const char *texto, const char *sub) que conte quantas vezes uma substring ocorre dentro de uma string maior.

138. Escreva uma função void `remove_espacos(char *s)` que remova todos os espaços em branco de uma string sem criar uma nova string (ou seja, alterando o conteúdo original).

139. Implemente uma função `void concatena(char *dest, const char *orig);` que concatene a string `orig` ao final de `dest`, sem usar `strcat`. Use apenas aritmética de ponteiros para percorrer e copiar os caracteres. No `main`, leia duas palavras, chame a função e exiba o resultado.

Bom estudo!