

Quiz 11 - Ponteiros em C/C++, Referência em Java/C# e Pilha/Fila/Lista Flexível

- Entrega 28 abr em 9:21
- Pontos 100
- Perguntas 10
- Disponível 28 abr em 8:50 - 28 abr em 9:21 31 minutos
- Limite de tempo Nenhum

Instruções

Este quiz aborda Ponteiros em C/C++, Referência em Java/C# e Pilha/Fila/Lista Flexível. Ele tem 10 questões de múltiplas escolha sendo que cada uma vale 10 pontos. Após o preenchimento de uma questão, o aluno não tem a opção de retorno à mesma. Este trabalho deve ser efetuado sem consulta.

Este teste foi travado 28 abr em 9:21.

Histórico de tentativas

	Tentativa	Tempo	Pontuação
MAIS RECENTE	Tentativa 1	12 minutos	80 de 100

Pontuação deste teste: 80 de 100

Enviado 28 abr em 9:20

Esta tentativa levou 12 minutos.



Pergunta 1


10 / 10 pts

Mostre a saída na tela para o código abaixo:

```
void fun(int *p)
{
    int q = 10;
    p = &q;
}

int main()
{
    int r = 20;
    int *p = &r;
    fun(p);
    printf("%d", *p);
    return 0;
}
```

- ☐ 10
- ☐ Compiler error
- Correto!
- ☒ 20
- ☐ Runtime Error

Basta executar o código. Esta questão foi obtida no www.geeksforgeeks.org  (<http://www.geeksforgeeks.org>).



Pergunta 2

10 / 10 pts

Um ponteiro é um tipo de dado cujo valor é o endereço de outro valor alocado em outra área da memória. Em relação aos ponteiros, avalie as afirmações a seguir.

I. Após a sequência de instruções abaixo, p armazena o endereço de x.

```
int *p;
int x = 5;
p = &x;
```

II. Referente a alocação dinâmica de memória em C, a função malloc usa o número de blocos de

memória que serão alocados na memória.

III. A saída na tela abaixo quando o usuário digita 15 como entrada de dados será 15 10

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

É correto o que se afirma em

☐ II, apenas.

☐ I, II e III.

Correto!

☒ I e III, apenas.

☐ II e III, apenas.

☐ I, apenas.

Execute os dois códigos. No caso da segunda afirmação, em C, o malloc aloca blocos de memória.



Pergunta 3

10 / 10 pts

Os ponteiros são variáveis que armazenam endereços de memória. Tal endereço pode ser de qualquer tipo de variável. Assim, temos ponteiros para int, double ou outros. Em relação aos ponteiros, avalie as afirmações a seguir.

I. Após a sequência de instruções abaixo, podemos afirmar que p[4] é igual a 3

```
int *p;
int v[]={10,7,2,6,3};
p = v;
```

II. A saída na tela abaixo quando o usuário digita 15 como entrada de dados será 5 15

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

III. A saída na tela abaixo será -1 3

```
int funcao (int a, int b, int *c){  
    *c = a-b;  
    return a*(b+1);  
}  
int main(){  
    int x = 1, y = 2, w = 3;  
    int z = funcao(x, y, &w);  
    printf("%d %d\n", w, z);  
    return 0;  
}
```

É correto o que se afirma em

- ☐ I, apenas.
- ☐ II e III. apenas.
- ☐ II, apenas.

Correto!

- ☒ I e III, apenas.
- ☐ I. II e III.

Execute os três códigos.



Pergunta 4

10 / 10 pts

Ponteiros são variáveis que guardam o endereço de memória. Em relação aos ponteiros, avalie as afirmações a seguir.

I. Após a sequência de instruções abaixo, podemos afirmar que *p é igual a 10

```
int *p;  
int v[]={10,7,2,6,3};  
p = v;
```

II. A saída na tela abaixo será -1 2 1

```
int funcao3 (int* a, int b){
    *a = *a-b;
    return *a+b;
}
int main(){
    int x = 1, y = 2;
    int z = funcao3(&x, y);
    printf("%d %d %d\n", x, y, z);
    return 0;
}
```

III. Dado o código abaixo, os números mínimo e máximo de acessos que precisam ser feitos para localizar um registro nessa estrutura contendo n elementos é 1 e n

```
typedef struct list {
    item_type item;
    struct list *next;
} list;
list *search_list(list *l, item_type x){
    return (l == NULL) ? NULL : ( (l->item == x) ? l : search_list(l->next, x));
}
```

É correto o que se afirma em

- ☐ I e III, apenas.
- ☐ II e III, apenas.
- ☐ I, apenas.
- ☐ II, apenas.

Correto!

- ☒ I, II e III.

Execute os três códigos.



Pergunta 5

10 / 10 pts

Assumindo que o tamanho do int é igual a 4 bytes, mostre a saída na tela para o código abaixo:

```
#define R 10
#define C 20

int main()
{
    int (*p)[R][C];
    printf("%d", (int)sizeof(*p));
    return 0;
}
```

☐ 200

Correto!

☒ 800

☐ 4

☐ 80

Basta executar o programa. Lembre-se que consideramos o int com 4 bytes. Esta questão foi obtida no www.geeksforgeeks.org.



Pergunta 6

10 / 10 pts

A fila é uma estrutura de dados onde o primeiro elemento que entra é o primeiro a sair. Essa estrutura é usada quando desejamos que a ordem de remoção dos elementos seja igual aquela em que eles foram inseridos em nossa estrutura. Considere o código abaixo pertencente a uma Fila contendo nó cabeça e os ponteiros primeiro e último.

```
public Fila metodo(){
    Fila resp = new Fila();
    Celula i = this.primeiro.prox;
    Celula j = i.prox;

    while (j != null) {
        resp.inserir(j.elemento);
        i.prox = j.prox;
        j.prox = null;
        if (i.prox != null) {
            i = i.prox;
            j = i.prox;
        } else
            j = null;
    }
    this.ultimo = i;
    return resp;
}
```

Considerando o código acima, avalie as afirmações a seguir.

- I. A execução do método mantém todos os elementos da fila inicial.
- II. O método apresentado funciona corretamente quando a fila está vazia.
- III. O objeto *resp* terá os elementos contidos nas posições pares de nossa fila, desconsiderando o nó cabeça.

É correto o que se afirma em

- ☐ II, apenas.
- ☐ I e III, apenas.
- ☒ III, apenas.
- ☐ I e II, apenas.
- ☐ I, II e III.

I. ERRADA - A execução do método faz com que a fila tenha somente os elementos que estavam nas posições ímpares da fila inicial, desconsiderando o nó cabeça.

II. ERRADA - Quando a fila estiver vazio, teremos um erro de execução no comando `Celula j = i.prox.`

III. CORRETA O objeto *resp* terá os elementos contidos nas posições pares de nossa fila, desconsiderando o nó cabeça.



Pergunta 7

10 / 10 pts

Uma lista encadeada é uma representação de uma sequência de registros na memória *Random Access Memory* (RAM) do computador. Cada elemento da sequência é armazenado em uma célula da lista sendo o primeiro elemento na primeira célula, o segundo na segunda, e assim por diante. Considere o código abaixo contendo um registro do tipo `NoLista` e uma função para inserir no início da lista (UFS'14 - adaptada).

```
struct NoLista {
    int elemento;
    struct NoLista *proximo;
};

struct NoLista * inserirInicio(struct NoLista *inicio, int num, int *erro){
    struct NoLista *novo;
    *erro = 0;
    novo = (struct NoLista*) malloc(sizeof(struct NoLista));
    if (novo == NULL){
        *erro = 1;
        return inicio;
    } else {
        novo->elemento=num;
        _____ /* (1) */
        return novo;
    }
}
```

Para que a função, que insere um novo elemento no início da lista e retorne o início da lista, funcione corretamente, a linha em branco, marcada com o comentário (1), deve ser preenchida com

☐ `inicio->proximo = novo;`

Correto!

- ☒ novo->proximo = inicio;
- ☐ inicio = novo;
- ☐ novo = inicio;
- ☐ novo->proximo = inicio->proximo;

O comando `/* (1) */` será `novo->proximo = inicio;`. Nesse caso, temos que o ponteiro novo aponta para a nova célula que contém o novo item e seu "próximo" aponta para o início atual. Quando a função for chamada, o novo valor de início deve ser atualizado com o endereço da célula recém criada que é retornado pela função em questão.



Pergunta 8

10 / 10 pts

Uma lista encadeada é uma estrutura de dados flexível composta por células sendo que cada célula tem um elemento e aponta para a próxima célula da lista. A última célula aponta para *null*. A lista encadeada tem dois ponteiros: primeiro e último. Eles apontam para a primeira e última célula da lista, respectivamente. A lista é dita duplamente encadeada quando cada célula tem um ponteiro anterior que aponta para a célula anterior. O ponteiro anterior da primeira célula aponta para *null*. A respeito das listas duplamente encadeadas, avalie as asserções a seguir.

I. Na função abaixo em C para remover a última célula, o comando `tmp = NULL` pode ser eliminado mantendo o funcionamento de nossa lista para outras operações. Contudo, isso pode gerar um vazamento de memória porque o espaço de memória relativo à célula ``removida" só será liberado no final da execução do programa.

```
int removerInicio() {
    if (primeiro == ultimo) errx(1, "Erro!");

    CelulaDupla *tmp = primeiro;
    primeiro = primeiro->prox;
    int elemento = primeiro->elemento;
    primeiro->ant = NULL;
    tmp->prox = NULL
    free(tmp);
    tmp = NULL;
    return elemento;
}
```

PORQUE

II. As linguagem C e C++ não possuem coleta automática de lixo como na linguagem JAVA. Essa coleta do JAVA é realiza pela Máquina Virtual Java que reivindica a memória ocupada por objetos que não são mais acessíveis.

A respeito dessas asserções, assinale a opção correta.

Correto!

- ☒ A asserção I é uma proposição falsa, e a asserção II é uma proposição verdadeira.
- ☐ As asserções I e II são proposições verdadeiras, mas a I não é uma justificativa correta da I.
- ☐ As asserções I e II são proposições falsas.
- ☐ As asserções I e II são proposições verdadeiras, e a II é uma justificativa correta da I.
- ☐ A asserção I é uma proposição verdadeira, e a asserção II é uma proposição falsa.

A primeira afirmação é falsa. Realmente, o comando `tmp = NULL` pode ser eliminado mantendo o funcionamento de nossa lista para outras operações. Isso, porque a variável `tmp` é uma variável local cujo escopo de vida limita-se a função em questão. A remoção do comando citado não causa qualquer vazamento de memória.

A segunda afirmação é verdadeira dado que as linguagem C e C++ não possuem coleta automática de lixo e o Java possui.



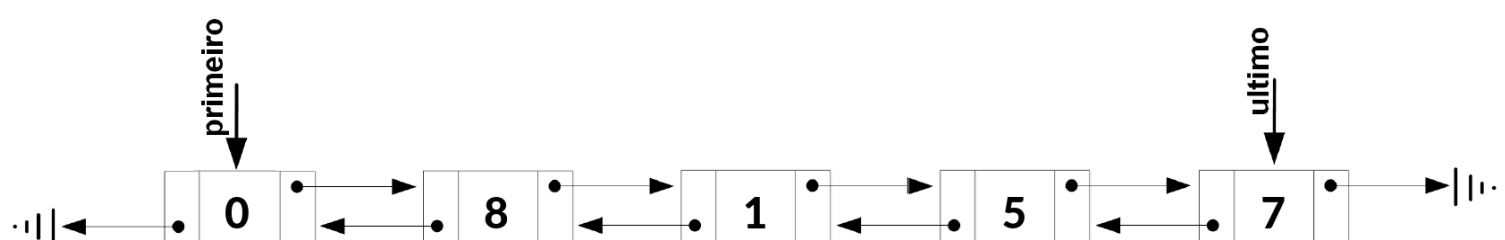
Pergunta 9

0 / 10 pts

Os conjuntos são tão fundamentais para a ciência da computação quanto o são para a matemática. Enquanto os conjuntos matemáticos são invariáveis, os conjuntos manipulados por algoritmos podem crescer, encolher ou sofrer outras mudanças ao longo do tempo. Chamamos tais conjuntos de conjuntos dinâmicos.

CORMEN T. H., et al., **Algoritmos: teoria e prática**, 3ed, Elsevier, 2012

A figura apresentada abaixo contém uma lista duplamente encadeada, um tipo de conjunto dinâmico.



O algoritmo abaixo manipula os elementos de uma lista duplamente encadeada.

```
Celula *i = primeiro->prox;
Celula *j = ultimo;
Celula *k;
while (j->prox != i){
    int tmp = i->elemento;
    i->elemento = j->elemento;
    j->elemento = tmp;
    i = i->prox;
    for (k = primeiro; k->prox != j; k = k->prox); j = k;
}
```

Considerando a figura e o código acima e seus conhecimentos sobre listas duplamente encadeadas, avalie as afirmações a seguir:

- I. A execução do algoritmo na lista faz com que o conteúdo da lista (da primeira para a última célula) seja: *0 7 5 1 8*.
- II. Sabendo que a primeira célula é o nó cabeça, o algoritmo inverte as células úteis quando temos um número ímpar dessas células.
- III. Na lista inicial, a execução do comando *primeiro->prox->prox->ant->elemento* exibe na tela o número 8.

É correto o que se afirma em

Você respondeu

☒ II, apenas.

Resposta correta

☐ I e III, apenas.

☐ III, apenas.

☐ I, II e III.

☐ I e II, apenas.

I. CORRETA. O algoritmo em questão inverte a ordem dos elementos. A ordem do primeiro elemento é mantida dado que ele é o nó cabeça.

II. ERRADA - A condição $i \neq j$ aborta o laço quando a quantidade de células é ímpar. A $j \rightarrow \text{prox} \neq$

i, quando essa quantidade é par.

III. CORRETA. A execução do comando primeiro->prox->prox->ant->elemento exibe na tela o número 8.



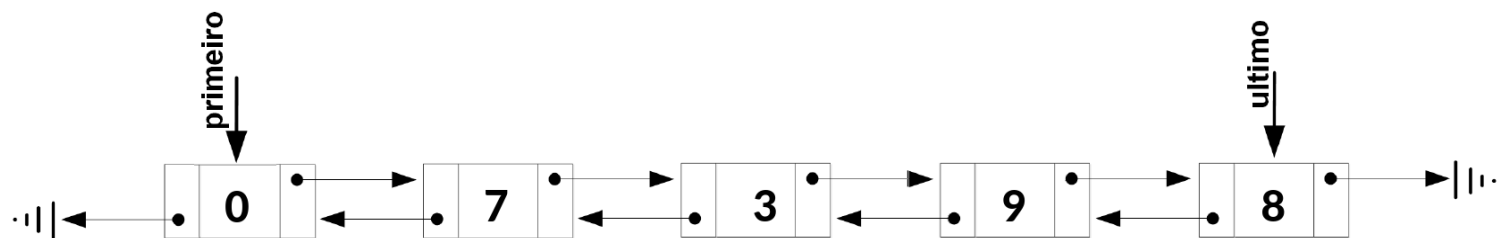
Pergunta 10

0 / 10 pts

Os conjuntos são tão fundamentais para a ciência da computação quanto o são para a matemática. Enquanto os conjuntos matemáticos são invariáveis, os conjuntos manipulados por algoritmos podem crescer, encolher ou sofrer outras mudanças ao longo do tempo. Chamamos tais conjuntos de conjuntos dinâmicos.

CORMEN T. H., et al., **Algoritmos: teoria e prática**, 3ed, Elsevier, 2012

A figura apresentada abaixo contém uma lista duplamente encadeada, um tipo de conjunto dinâmico.



O algoritmo abaixo é manipula os elementos de uma lista duplamente encadeada.

```

Celula *i = primeiro;
Celula *j = ultimo;
Celula *k;
while (i != j){
    int tmp = i->elemento;
    i->elemento = j->elemento;
    j->elemento = tmp;
    i = i->prox;
    for (k = primeiro; k->prox != j; k = k->prox); j = k;
}
  
```

Considerando a figura e o código acima e seus conhecimentos sobre listas duplamente encadeadas, avalie as afirmações a seguir:

- I. Na lista inicial, a execução do comando primeiro->prox->prox->prox->prox->ant->elemento exibe na tela o número 8.
- II. A execução do algoritmo na lista faz com que o conteúdo da lista (da primeira para a última célula) seja: 8 9 3 7 0.
- III. O algoritmo apresentado funciona corretamente quando nossa lista tem um número ímpar de células.

É correto o que se afirma em

☐ I e II, apenas.

☐ I, II e III.

Resposta correta

☐ II, apenas.

Você respondeu

☒ III, apenas.

☐ I e III, apenas.

I. ERRADA - A condição $i \neq j$ aborta o laço quando a quantidade de células é ímpar. A $j \rightarrow \text{prox} \neq i$, quando essa quantidade é par.

II CORRETA. O algoritmo em questão inverte a ordem dos elementos. A ordem do primeiro elemento é mantida dado que ele é o nó cabeça.

III. ERRADA. A execução do comando primeiro->prox->prox->prox->prox->ant->elemento exibe na tela o número 9.

Pontuação do teste: 80 de 100