1)

A linguagem de programação C permite que os parâmetros sejam passados para as funções de duas maneiras, por valor e por referência. Na passagem por valor, uma expressão pode ser utilizada na chamada. Na passagem por referência, o endereço de uma variável deve ser passado na chamada da função. Dessa forma, a função pode modificar a variável diretamente, o que em geral não é recomendável, mas há situações onde esse recurso é necessário. Sempre que possível é recomendável utilizar a forma de passagem por valor, para evitar "efeitos colaterais", mas em algumas situações, esses efeitos são desejáveis quando é preciso criar uma função que retorne mais de um valor.

 Fonte:Disponível em:< https://www.ime.usp.br/~elo/IntroducaoComputacao/Funcoes%20passagem%20de%20parametros.htm>Acesso.13.Ago.2018.

  Como exemplo um número complexo, que possui uma parte real e outra complexa. Analise o trecho de programa e as asserções a seguir:

#include <stdio.h>

#include <stdlib.h>

void complexo2 (float \*r, float \*t);

int main ()

{

float a, b;

printf ("Entre com um numero complexo (2 numeros inteiros): ");

scanf("%f %f", &a, &b);

complexo2 ( &a, &b);

printf("O quadrado do número complexo é %f + i %f\n", a, b);

system("pause");

return 0;

}

Neste contexto, julgue as afirmações que se seguem.

 I. No main são declaradas apenas duas variáveis reais a e b. E tanto na chamada da função scanf, quanto na chamada da função complexo2, as variáveis a e b são precedidas pelo caractere '\*'.

  II. Esse caractere ‘\*’ indica a necessidade de que esses parâmetros sejam modificados pela função e retornem os valores apropriados.

 III. No caso da função scanf, esses valores correspondem aos valores digitados pelo usuário, mas no caso da função complexo2, esses valores são computados pela função. Ou seja, os parâmetros a e b são utilizados apenas para saída, e no caso da complexo2, os parâmetros a e b são utilizados como entrada e saída da função.

 IV. Os próprios valores armazenados nas variáveis a e b são utilizados como entrada da função complexo2, ou seja, como parte real e imaginária para calcular o valor do quadrado do número complexo, e ao mesmo tempo para devolver o resultado.

  V. Na função printf, as variáveis a e b não são precedidas pelo caractere '&', portanto, não são modificadas pela função printf.

  VI. Quando deseja-se criar uma função que devolva mais de um valor, é preciso definir esses parâmetros com um caractere '&' no protótipo da função, e ao chamar a função, os parâmetros utilizados na chamada correspondentes à entrada precisam ser precedidos pelo caractere '\*' para indicar que eles podem ser modificados pela função chamada.

É correto apenas o que se afirma em:

**Alternativas:**

* a)

I, II e VI.

* b)

III, IV e V.

Alternativa assinalada

* c)

I e IV.

* d)

II e V.

* e)

III e VI.

2)

Leia o trecho de programa a seguir:

#include<stdio.h>

int x = 10;

int main(){

int x = -1;   int b;

{

extern int x;     b = x;

}

printf("\n Valor de x = %d",x);

printf("\n Valor de b (x global) = %d",b);

return 0;

}

De acordo com o trecho de programa, analise as asserções a seguir:

 I. Na linguagem C, para acessar o valor de uma variável global dentro de uma função que possui uma variável local com mesmo nome, deve-se usar a instrução extern.

**PORQUE**

 II. Utilizar variáveis globais e locais com mesmo nome na linguagem C necessita criar uma nova variável chamada “b”, com um bloco de instruções, que atribui à nova variável o valor “interno” de x.

A respeito dessas asserções, assinale a alternativa correta:

**Alternativas:**

* a)

As asserções I e II são proposições verdadeiras, e a II é uma justificativa da I.

* b)

As asserções I e II são proposições verdadeiras, mas a II não é uma justificativa da I.

* c)

A asserção I é uma proposição verdadeira, e a II é uma proposição falsa.

Alternativa assinalada

* d)

A asserção I é uma proposição falsa, e a II é uma proposição verdadeira.

* e)

As asserções I e II são proposições falsas.

3)

Para usar de forma mais otimizada a memória, existe uma alternativa chamada recursividade em cauda. Nesse tipo de técnica a recursividade funcionará como uma função iterativa. Uma função é caracterizada como recursiva em cauda quando a chamada a si mesmo é a última operação a ser feita no corpo da função. Nesse tipo de função, o caso base costuma ser informado como parâmetro, o que resultará um comportamento diferente.

A principal vantagem da recursividade em cauda consiste em:

**Alternativas:**

* a)

as instâncias vão sendo criadas, porém quando chega na última, as funções precisam retornar o valor para “quem” invocou, gerando otimização na memória, pois precisa armazenar um ponto para devolução de valores.

* b)

as instâncias vão sendo criadas, porém quando chega na última, as funções não precisam retornar o valor para “quem” invocou, gerando otimização na memória, pois não precisa armazenar nenhum ponto para devolução de valores.

Alternativa assinalada

* c)

as instâncias vão sendo descartadas, porém quando chega na última, as funções não descartam o valor invocado, gerando otimização na memória, pois não precisa armazenar nenhum ponto para devolução de valores.

* d)

as instâncias vão sendo criadas, porém quando chega na última, as funções não retornam valor, gerando conflito na memória, pois não precisa armazenar nenhum ponto para devolução de valores.

* e)

as instâncias vão sendo criadas, porém quando chega na última, as funções não precisam retornar o valor para “quem” invocou, gerando conflito na memória, pois não precisa armazenar nenhum ponto para devolução de valores.

4)

“Muitos problemas têm a seguinte propriedade: cada instância do problema contém uma instância menor do mesmo problema. Diz-se que esses problemas têm estrutura recursiva. Para resolver tal problema, pode-se aplicar o seguinte método: se a instância em questão for pequena, resolva-a diretamente; senão, reduza-a a uma instância menor do mesmo problema, aplique o método à instância menor, volte à instância original. A aplicação desse método produz um algoritmo recursivo.” (FEOFILOFF, 2017, p. 1). Baseado nesse conceito, avalie as asserções a seguir:

 I -  Recursividade significa indicar quando um problema maior pode ser dividido em instâncias menores do mesmo problema.

**PORQUE**

 II - A técnica de recursividade pode substituir o uso de estruturas de repetição tornando o código mais elegante.

A respeito dessas asserções, assinale a alternativa correta:

**Alternativas:**

* a)

As asserções I e II são proposições verdadeiras, e a II é uma justificativa da I.

Alternativa assinalada

* b)

As asserções I e II são proposições verdadeiras, mas a II não é uma justificativa da I.

* c)

A asserção I é uma proposição verdadeira, e a II é uma proposição falsa.

* d)

A asserção I é uma proposição falsa, e a II é uma proposição verdadeira.

* e)

As asserções I e II são proposições falsas.

5)

Recursividade é uma técnica sofisticada em programação, na qual uma função chama a si mesma criando várias instâncias (chamadas recursivas). Embora seja uma técnica que proporciona um código mais limpo e facilita a manutenção, seu uso deve levar em consideração a quantidade de memória necessária para a execução do programa. Nesse contexto, avalie as asserções a seguir:

I. A cada chamada recursiva é alocado recursos na memória para a função, se a função for muito grande poderá ocorrer um acúmulo de memória.

**PORQUE**

 II. É preciso avaliar o custo-benefício em se ter um código mais sofisticado em detrimento de uma estrutura de repetição, pois a segunda opção gasta menos memória.

A respeito dessas asserções, assinale a alternativa correta:

**Alternativas:**

* a)

As asserções I e II são proposições verdadeiras, e a II é uma justificativa da I.

* b)

As asserções I e II são proposições verdadeiras, mas a II não é uma justificativa da I.

* c)

A asserção I é uma proposição verdadeira, e a II é uma proposição falsa.

* d)

A asserção I é uma proposição falsa, e a II é uma proposição verdadeira.

Alternativa assinalada

* e)

As asserções I e II são proposições falsas.