Nome: Eduardo Lucas Lemes Januário CT3037568
Nome: Guilherme Batista de Souza CT3037274

Revisão para Prova

Instruções: Responda as questões de forma objetiva e fundamentada. Utilize exemplos quando necessário para justificar suas respostas.

1. Ponteiros e Alocação de Memória
Explique o conceito de ponteiros em C. Como a alocação dinâmica de memória utilizando calloc() e free() (pode melhorar a eficiência do uso da memória em um programa? Dê um exemplo de código para ilustrar sua resposta.

Um ponteiro é uma variavel que armazena o endereço de memória de outra variavel como valor. A alocação dinamica da memoria permite reservar uma quantidade de memoria durante a execução do programa, ao invés de definir um tamanho fixo. A partir do Calloc() alocamos memoria e iniciamos todos os bytes com zero, e com free() podemos liberar a memoria que nao esta sendo utilizada, evitando vazamento.

```
#include <stdio.h>
#include <stdlib.h>

int main () {
        int tamanho = 5;
        int * vetor = (int *) calloc(tamanho, sizeof(int));
        if (vetor != NULL) {
                for (int i=0; i<tamanho; i++){
                        vetor[i] = i*2;
                }
        }
        free(vetor);
        return 0;
}
```

2. Vetores e Ponteiros
Considere o seguinte trecho de código em C:

```
#include <stdio.h>

void modificaVetor(int *v, int tamanho) {
        for (int i = 0; i < tamanho; i++) {
                *(v + i) *= 2;
        }
}
```

```
int main() {
	int vetor[] = {1, 2, 3, 4, 5};
	int tamanho = sizeof(vetor) / sizeof(vetor[0]);

	modificaVetor(vetor, tamanho);

	for (int i = 0; i < tamanho; i++) {
		printf("%d ", vetor[i]);
	}
	return 0;
}
```

Explique o que acontece com os valores do vetor após a execução do programa. Qual a importância do uso de ponteiros na passagem de vetores para funções?

Após a execução do programa cada um dos valores do vetor serão multiplicados por 2 e impresos. A importancia do uso de ponteiros na passagem de vetores para funções é a possibilidade da alteração do vetor de forma direta, tornando o codigo mais eficiente, ja que nao é necessario retornar um novo vetor.

3. Matrizes e Acesso à Memória

Dado o seguinte código em C:

```
int matriz[3][3] = { {1, 2, 3}, {4, 5, 6}, {7, 8, 9} };
printf("%d", *(*(matriz + 1) + 2));
```

Qual será a saída do programa? Explique o raciocínio por trás do acesso ao elemento da matriz usando aritmética de ponteiros.

A saida do programa será 6.
1º Matriz é um ponteiro para a primeira linha da matriz.
2º *(Matriz + 1) acessa o primeiro elemento da segunda linha da matriz.
3º *(Matriz + 1) + 2 desloca esse ponteiro para a terceira coluna da segunda linha da matriz.
4º *(*(Matriz+1)+2) acessa o valor armazenado nessa posição, que é 6.

4. Funções e Passagem de Parâmetros

Qual a diferença entre passagem de parâmetros por valor e por referência em C? Em que situações é mais vantajoso utilizar cada uma delas? Escreva um exemplo de função que utilize a passagem por referência.

A passagem de parametros por valor passa uma copia do valor que esta armazenado na memoria, neste caso se voce modificar este valor, ele nao modificara o valor original, a variavel original; A passagem de parametros por referencia passa o endereço damemoria, o que esta alocado neste endereço, ou seja, voce trata o valor bruto, o valor original, sendo assim quaisquer alterações realizadas na variavel e salvas não poderão ser revertidos.

Utiliza-se a passagem de parametros por valor para dados pequenos, como int oou float, pois não precisamos nos preocupar muito em criar copias do mesmo, ou quando voce necessariamentew precisa tratar da copia de uma dado. A passagem de parâmetros por referencia é utilizada para guardar dados, como vetores e matrizes, tendo em vista que seria custoso e trabalhoso fazer copias das mesmas, e tambem é benfico a elas trabalhar diretamente com o endereço da memoria de determinada variavel.

```
#include <stdio.h>

void alterarReferencia(int * x){
        *x=20;
}

int main () {
        int a = 10;
        alterarReferencia(&a);
        printf("%d", a);
        return 0;
}
```

5. Recursividade – Função de Ackermann

A função de Ackermann é um exemplo clássico de recursividade, definida da seguinte forma:

```
            n+1,                    se m=0
a(m,n) =    a(m -1, 1),             se m>0 e n=0
            a(m-1, a(m,n-1)),       se m>0 e n>0
```

Implemente essa função em C e explique como a recursão funciona nesse caso.

```
a(int m, int n){
        if(n==0){
                return n+1;
        }else if(m>0 && n==0){
                return a(m-1, 1;)
        }else{
                return a(n-1, a(m, n-1));
        }
}
```

6. Operações Fundamentais e Derivadas em Cadeias de Caracteres

Explique quais são as operações fundamentais e derivadas em cadeias de caracteres.

Existem três operações fundamentais, o primeiro que fornece como resultado o primeiro caractere da cadeia em questão; o restante que fornece como resultado uma nova cadeia que é iguel a própria cadeia anterior, apenas removendo o primeiro caractere; concatenação que fornece como resultado uma nova cadeia que é a junção de duas ou mais cadeias escolhidas e passadas como parametros.

Existem diversas operações derivadas, vamos comentar acerca de algumas, o comprimento que fornece como resultado o numero de caracteres da cadeia em questão; a subcadeia que fornece como resultado uma nova cadeia que é igual a parte da cadeia em questão; inserção que fornece como resultado uma nova cadeia em que a cadeia y é inserida na cadeia x, a partir da sua i-ésima posição; a remoção que fornece como resultado uma nova cadeia que é copia de x, exceto pela parte correspondente a subcadeia (x, i ,n) que foi removida.