DAINF-UTFPR/Pato Branco

# Simulado da 1a Avaliação

Algoritmos e Estrutura de Dados I (AE42CP)

1. Caso seja feita alocação dinâmica de memória dentro de uma função, cite um cuidado que devemos tratar e o porquê.
2. O que é um Tipo Abstrato de Dados (TAD) e qual a característica fundamental na sua utiliza- ção?
3. Quais as vantagens de se programar com TADs?
4. Identifique os possíveis erros nos códigos e explique por que tais trechos de código estão errados. Em seguida, apresente uma solução para cada problema.

|  |  |  |
| --- | --- | --- |
| a) - | 1. | \*float multiplicar(float \*f1, float \*f2){ |
|  | 2. | float \*p; |
|  | 3. | p = (f1) \* (f2); |
|  | 4. | return 0; |
|  | 5. | } |
|  | 6. | int main(){ |
|  | 7. | int a = 10, b = 5; |
|  | 8. | int \*c = multiplicar(a, b); |
|  | 9. | return 0; |
|  | 10. | } |
| b) - | 1. | typedef struct P3D{ |
|  | 2. | int x, y, z; |
|  | 3. | }P3D; |
|  | 4. | typedef struct Ponto3D{ |
|  | 5. | int x, y, z; |
|  | 6. | }Ponto3D; |
|  | 7. | int main(){ |
|  | 8. | Ponto3D a = 1 2 3; |
|  | 9. | Ponto3D b; |
|  | 10. | P3D c = 3 2 1; |
|  | 11. | P3D \*p1 = &a; |
|  | 12. | Ponto3D \*p2 = &c; |
|  | 13. | b = a; |
|  | 14. | return 0; |
|  | 15. | } |

1. Implemente uma função que receba uma pilha estática de números inteiros. A função deverá reorganizar a pilha de forma que nenhum item esteja empilhado sobre um outro menor, ou seja, a pilha deve ser ordenada.
2. Implemente uma função que receba uma string. A função deverá verificar, utilizando uma pilha estática, se a string é um palíndromo.
3. Qual a diferença entre *struct* e *union*? Dê exemplos.
4. Identifique e corrija os erros (se houver) dos fragmentos de código abaixo:
   1. - struct Musica{

char genero[16]; char titulo[101]; char artista[51];

unsined int duracao; //segundos unsined int tamanho; //KB

}

int main(){ Musica m;

strcpy(m->genero, "Sertanejo"); strcpy(m->, "Evidencias");

strcpy(m->artista, "Chitaozinho e Xororo"); m->duracao = 180;

m->tamanho = 2048;

}

* 1. - int x = 10; int \*p;

p = &x;

\*p = x + ’a’;

(\*p)++;

free(p);

* 1. - float \*p = (int\*) calloc(sizeof(float));

\*p = 10;

1. Simule o uso de pilha para conversão da seguinte expressão na notação infixa para a notação pós-fixa (polonesa reversa): *a* + *b* + *c* (*b d* + *e* + *a*)*/c d a*. Essa simulação pode ser feita utilizando uma tabela, conforme a sugestão do exemplo abaixo:

∗ − − −

Conversão de infixa para pós-fixa Exemplo: A - B \* C + D

|  |  |  |
| --- | --- | --- |
| **Entrada** | **Pilha** | **Saída** |
| A |  | A |
| - | - | A |
| B | - | AB |
| \* | - \* | AB |
| C | - \* | ABC |
| + | + | ABC\*- |
| D | + | ABC\*-D |
|  |  | ABC\*-D+ |

1. O Dr. Hanz Chukrutz é um geneticista entusiasta da área de computação, especialmente estru- tura de dados. Em um dia desses ele decidiu utilizar listas estáticas para representar sequenci- amento de DNA. Nesse tipo de lista estática, cada elemento é um caractere representando um nucleotídeo. Em um estudo sobre doenças genéticas, em cada lista estática, uma subsequência é procurada. Para ajudar o Dr. Chukrutz em sua pesquisa, implemente uma função que receba duas listas estáticas de caracteres l1 e l2, que representam uma sequência e uma subsequência, respectivamente. Essa função deverá retornar 1, caso seja possível encontrar a subsequência l2 dentro da l1. Exemplo:

* Entrada:

**–** l1 = {‘C’, ‘G’, ‘U’, ‘A’, ‘C’, ‘G’, ‘G’, ‘A’, ‘G’, ‘C’, ‘U’, ‘U’, ‘C’, ‘G’, ‘G’ , ‘A’, ‘G’, ‘C’, ‘C’}

**–** l2 = {‘U’, ‘U’, ‘C’, ‘G’, ‘G’}

* Saída: 1, já que a subsequência apresentada em l2 existe em l1.

# Anexos

Estruturas e protótipos de funções que podem ser utilizados na resolução de exercícios de implementação.

#define TMAX 200;

typedef struct{ int item[TMAX]; int tam;

}Lista;

typedef struct{ int item[TMAX]; int topo;

}Pilha;

typedef struct{ int item[TMAX];

int tam, ini, fim;

}Fila;

Lista \*criar\_lista(); Pilha \*criar\_pilha(); Fila \*criar\_fila();

int lista\_cheia(Lista \*l); int lista\_vazia(Lista \*l); int pilha\_cheia(Pilha \*p); int pilha\_vazia(Pilha \*p); int fila\_cheia(Fila \*f); int fila\_vazia(Fila \*f);

void inserir(Lista \*l, int chave); // insere a chave na última posição da lista

void empilhar(Pilha \*p, int chave); int desempilhar(Pilha \*p);

int obter\_topo(Pilha \*p); // retorna o valor que está no topo da pilha sem desempilhar

void enfileirar(Fila \*p, int chave); ind desenfileirar(Fila \*p);

void liberar\_lista(Lista \*l); void liberar\_pilha(Pilha \*p);

void liberar\_fila(Fila \*f);

**Obs.: não é necessária a implementação dos protótipos acima.**