Universidade de Itaúna

Curso: Ciência da Computação

Disciplina: Algoritmos e Estrutura de Dados II

Professor: Adriano Benigno Moreira

Aluno (a): Davi Ventura Cardoso Perdigão

Data: 21/10/2020

II Avaliação

1 - A) A struct utilizada nesses processos, na grande maioria dos casos, incluem variáveis para o armazenamento de valores (int, float) e ponteiros que irão auxiliar no processo de inicialização e finalização dessas estruturas.

```
typedef struct ponto
{
    int x,y;
    struct ponto *proximo;
}t_ponto;
```

B) Geralmente sua principal função é armazenar o endereço da memória do primeiro elemento ou do próximo elemento, o que possibilita a inicialização e auxilia no funcionamento das estruturas citadas.

C) Primeiro é necessário a criação de um ponteiro (ini ponto), já que a função MALLOC irá nos retornar um endereço de memória. Logo após, é necessários usar o cast (t ponto *) para mudar o tipo de ponteiro que será retornado por essa função. E, por fim, como parâmetro para a função MALLOC iremos utilizar o comando SIZEOF, responsável por informar a quantidade de bytes que nosso tipo de dado ocupa na memória. A combinação de todos esses elementos é o que possibilita a utilização da alocação dinâmica, claro, existem outros comandos, porém esta é a estrutura básica da alocação.

```
ini_ponto= (t_ponto *) malloc(sizeof(t_ponto));//ALOCAR E APONTAR O ENDEREÇO DA MEMORIA DE t_ponto PARA ini_ponto;
```

2 - A) As funções que a pilhar deve conter para a sua aplicação e funcionamento são:

- Inicialização
- Inserção de novos elementos
- Eliminar elementos da pilha
- Exibição da Pilha
- Recuperação de elementos

B) Essa função é responsável por declarar um ponteiro que aponta para o primeiro elemento da pilha, exibe o dado armazenado ali, pega o endereço do próximo nó, exibe o que está armazenado nele também, e assim se segue, até o fim da pilha. Além de exibir para os usuários os valores presentes em cada nó.

C) int empilhar (Pilha * monte, char *dado){

```
Elemento *novo_elemento;
if ((novo_elemento = (Elemento *) malloc (sizeof (Elemento))) == NULL)
return -1;
if ((novo_elemento->dado = (char *) malloc (50 * sizeof (char)))
== NULL)
return -1;
strcpy (novo_elemento->dado, dado);
novo_elemento->próximo = monte->início;
monte-> início = novo_elemento;
monte->tamanho++;
}
```

3 - A) Como a fila “se movimenta” permitindo a manipulação dos dados nos dois extremos, e o vetor é uma estrutura estática, ao implementar um mecanismo circular (fila circular) podemos aproveitar o máximo do vetor. Com a fila estática circular, reaproveitamos os espaços liberados com a remoção do início da fila, e também sabemos se a fila está cheia ou vazia, pois, se inserimos todas as posições da fila e depois removemos todos, não existe mais elementos, porém não temos como inserir novos.

B) Elas servem como ponto de controle, pois a base de uma fila é o FIFO (First in, First out / Primeiro a entrar, primeiro a sair) e armazenar essas posições possibilita o trabalho e a adição de elementos na fila.

C) void fila_sair() //Retirar o primeiro elemento da Fila

```
{
if (fila.ini==fila.fim)
  {
  printf("\nFila vazia, aguenta ai que logo aparecerá alguém!\n\n");
  system("pause");
  }
```

```
  else
{
int i;
for (i = 0; i < tamanho; i++)
   {
   fila.dados[i] = fila.dados[i+1];
            }
fila.dados[fila.fim] = 0;
fila.fim--;
}
}
```

4 - A) Uma Lista deve ter em sua composição variáveis do tipo ponteiro, pois para que seja encontrado e incluído dados no meio, início e fim, é necessário o ponteiro devido a sua capacidade de armazenar endereços.

B) Sim, é possível inserir um novo elemento, sem a necessidade de ser somente no início ou no final. Podemos também inserir um novo elemento no meio da fila. Basta criar uma nova célula, criar um ponteiro, apontar essa nova célula para um elemento existente no meio da lista, pedir para o anterior apontar para o elemento criado. Sendo assim uma inserção no meio da lista. Mas sempre devemos pedir para a célula apontar primeiro e depois ser apontada pelo elemento existente para que a lista não se quebre.

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

typedef struct no{
   int num;
   struct no *proximo;
}No;

No* criar_no(void)
  {
     No *novo;
```

```
        novo = (No *) malloc(sizeof(No));
        //No *novo = (No *) malloc(sizeof(No));
        if (novo == NULL)
          {
             printf("Memoria insuficiente \n");
             exit(2);
          }
        return novo;
    }
    void
insere (int x, celula *p)
{
    celula *nova;
    nova = malloc (sizeof (celula));
    nova->conteudo = x;
    nova->prox = p->prox;
    p->prox = nova;
}
No* inserir_no_inicio (No* Lista, int dado)
{
     No *novo_no=criar_no();
     novo_no->num=dado;
     if(Lista == NULL)
       {
          Lista=novo_no;
          novo_no->proximo=NULL;
       }
     else
       {
       novo_no->proximo=Lista;
```

```
      Lista=novo_no;
      }
    return Lista;
}
void imprimir_lista (No* Lista)
{
    No *aux = Lista;
    while (aux !=NULL)
      {
          printf("\nElemento encontrado: %i ",aux->num);
          aux=aux->proximo;
      }
}
void pesquisa_lista (No* Lista)
{
    No *aux = Lista;
    int num_aux, local=0;
    printf("\nInforme o valor a ser pesquisado: ");
    scanf("%i",&num_aux);
    while (aux !=NULL)
      {
          if (num_aux == aux->num)
            {
                printf("\nElemento encontrado: %i \n",aux->num);
                local=1;
            }
          aux=aux->proximo;
      }
    if (local==0)
      {
```

```
      printf("\nElemento nao encontrado na lista\n");
      }
}
void apaga_lista (No* Lista)
{
    No *aux = Lista;
    while (aux !=NULL)
       {
          No *temp = aux->proximo;
          free(aux);
          aux=temp;
       }
}
int main (void)
{
    No *Lista=NULL;
    int op=1, num_aux;
    while(op==1)
      {
          printf("Digite o numero: ");
          scanf("%i", &num_aux);
          void insere_final(No** lista, int v)
{
No *n, *aux;
n = (No*) malloc(sizeof(No));
n>valor = v;
n>prox = NULL; // Ultimo elemento da lista
if (*lista == NULL)
{
*lista = n;
```

```
}
Else
      {
       aux = *lista;
       while (aux>    prox != NULL)
   {
                      aux = aux>prox;
}
      aux> prox = n;
      }
}
typedef struct no
{
int valor;  //Vai armazenar o valor informado
struct no* prox; // Irá possibilitar o inicio da lista, armazenando a posição dos nós.
}No;


          Lista=inserir_no_inicio(Lista,num_aux);
          printf("Deseja digitar outro numero? <1>Sim <2>Nao ");
        scanf("%i", &op);
      }
    imprimir_lista(Lista);
    pesquisa_lista(Lista);
    apaga_lista(Lista);
    free(Lista);
    system("pause");
}
```

5 - A) void insere_final(No** lista, int v)

```
{
No *n, *aux;
n = (No*) malloc(sizeof(No));
n>valor = v;
n>prox = NULL; // Ultimo elemento da lista
if (*lista == NULL)
{
*lista = n;
}
Else
      {
      aux = *lista;
      while (aux>       prox != NULL)
         {
                  aux = aux>prox;
}
     aux> prox = n;
     }
}
typedef struct no
{
int valor;  //Vai armazenar o valor informado
struct no* prox; // Irá possibilitar o início da lista, armazenando a posição dos nós.
}No;
```

B) int remove(No** lista)

```
 {
No *n, *aux;
int v;
if (*lista == NULL)
        {
return 1;
}
if ((*lista)>prox == NULL)
{
n = *lista;
*lista = NULL;
}
}
```