Aula 01 - Revisão de C

Estrutura de dados e seus algoritmos

Profa: Taiane C. Ramos

Conteúdo programático

Ponteiros

Vetores

Alocação dinâmica de memória

Tipos Estruturados de Dados

Listas Encadeadas

Recursão

Listas

Pilhas

Filas

Ponteiros

int a; -> inteiro

int *p; -> ponteiro para inteiro

p = &a; -> p recebe o endereço de a

*p = 6; -> o conteúdo apontado por p recebe 6

printf(“%p”, &p); -> imprime o endereço de p

printf(“%p”, p); -> imprime o endereço de a

printd(“%d”, *p); -> imprime o conteúdo de a;

1	a	int
2	p	endereço de uma var int

Ponteiros

```
int main(void){
        int a, *p;
        a = 5;
        p = &a;
        *p = 6;
        printf("endereço de p: %p \n", &p);
        printf("endereço de a:  %p \n", &a);
        printf("endereço que está em p: %p \n", p);
        printf("conteúdo da variável apontada por p: %d \n", *p);
        printf("conteúdo de a: %d \n", a);
        return 0;
}
```

Passagem de parâmetros

- Por valor
  - É passado o conteúdo da variável para uma variável interna da função
  - O conteúdo da variável não é alterado fora da função
- Por referência
  - É passado o endereço da variável original
  - O conteúdo da variável será alterado fora da função

Passagem de parâmetros por valor

```
#include <stdio.h>

void troca (int px, int py ) {
        int temp;
        temp = px;
        px = py;
        py = temp;
}
```

```
int main ( void ) {
        int a = 5, b = 7;
        troca(a, b);
        printf("%d %d \n", a, b);
        return 0;
}
```

Passagem de parâmetros por referência

```
#include <stdio.h>
void troca (int *px, int *py ) {
        int temp;
        temp = *px;
        *px = *py;
        *py = temp;
}
```

```
int main ( void ) {
        int a = 5, b = 7;
        troca(&a, &b);
        printf("%d %d \n", a, b);
        return 0;
}
```

Vetores

- Uma alocação contígua de memória de n variáveis do mesmo tipo.
- Declaração
  - int v[10];
- Primeira posição de v
  - v[0];
- Última posição de v
  - v[9];
- Memória não alocada
  - v[10];

Vetores com alocação estática

```
#include <stdio.h>

int main ( void ) {
        int v[10];
        int i;
        for(i = 0; i < 10; i++)
                scanf(“%d”, &v[i]);
        for(i = 0; i < 10; i++)
                printf(“%d ”, v[i]);
        return 0;
}
```

Vetor como parâmetro para função

```
#include <stdio.h>
void incr_vetor ( int n, int *v ) {
        int i;
        for (i = 0; i < n; i++)
                v[i]++;
}
```

```
int main ( void ) {
        int a[ ] = {1, 3, 5};
        incr_vetor(3, a); \\o vetor é um ponteiro
        printf("%d %d %d \n", a[0], a[1], a[2]);
        return 0;
}
```

Vetores com alocação dinâmica

```
#include <stdlib.h>

int main(void){
        int *v;
        v = (int *) malloc(10 * sizeof(int));
        return 0;
}
```

Exercício: Ler n floats e printar a média

Roteiro:

//Ler a quantidade n

//Alocar um vetor dinamicamente

//Ler os n valores

//Passar o vetor pra uma função que calcula a média

//Imprimir a média

//Desalocar a memória

Exercício: Ler n floats e printar a média

```
#include <stdio.h>

#include <stdlib.h>

float media(float n, float* v){

        float sum = 0;

        float media = 0;

        for(int i = 0; i < n; i++){

                sum = sum + v[i];

        }

        media = sum / n;

        return media;

}
```

```
int main(void){

        float* v;

        float n, res;

        scanf("%f", &n);

        v = (float*) malloc(n * sizeof(float));

        for(int i = 0; i < n; i++){

                scanf("%f", &v[i]);

        }

        res = media(n, v);

        printf("%.1f \n", res);

        free(v);

        return 0;

}
```

Estruturas: struct

- Uma estrutura para representar um ponto 2d no espaço.

```
struct ponto2d{
    float x;
    float y;
};
```

Exemplo: Lê e imprime as coordenadas x e y de um ponto

```
#include <stdio.h>

struct ponto2d {
        float x;
        float y;
};

int main (void) {
        struct ponto2d p;
        printf("Digite as coordenadas do ponto(x y): ");
        scanf("%f %f", &p.x, &p.y);
        printf("O ponto fornecido foi: (%.2f,%.2f)\n", p.x, p.y);
        return 0;
}
```

Modificar o exemplo

- Usar typedef na struct
- Usar ponteiro para struct
- Alocar a struct dinamicamente

Exemplo: Lê e imprime as coordenadas x e y de um ponto

```
#include <stdio.h>

typedef struct ponto2d {
        float x;
        float y;
} Ponto2d;
```

```
int main (void) {
        struct ponto2d p;
        printf("Digite as coordenadas do ponto(x y): ");
        scanf("%f %f", &p.x, &p.y);
        printf("O ponto fornecido foi: (%.2f,%.2f)\n", p.x, p.y);
        return 0;
}
```

Exemplo: Lê e imprime as coordenadas x e y de um ponto

```
#include <stdio.h>
#include <stdlib.h>
typedef struct ponto2d {
        float x;
        float y;
} Ponto2d;
```

```
int main (void) {
        Ponto2d *p;
        p = (Ponto2d*) malloc (sizeof(Ponto2d));
        printf("Digite as coordenadas do ponto(x y): ");
        scanf("%f %f", &p->x, &p->y);
        printf("O ponto fornecido foi: (%.2f,%.2f)\n", p->x, p->y);
        return 0;
}
```

Listas - Struct

```
typedef struct lista {
        int info;
        struct lista* prox;
} TLista;
```

Listas - Cria lista

```
TLista* cria_lista (void) {
        return NULL;
}
```

Listas - Insere um elemento na cabeça da lista

```
TLista* insere_inicio (TLista* li, int i) {
    TLista* novo = (TLista*) malloc(sizeof(TLista));
    novo->info = i;
    novo->prox = li;
    return novo;
}
```

Listas - Imprime lista

```
void imprime_lista (TLista* li) {
        TLista* p;
        for (p = li; p != NULL; p = p->prox){
                printf("info = %d \n", p->info);
        }
}
```

Listas - Insere no final

```
TLista* insere_fim (TLista* li, int i) {
        TLista* novo = (TLista*) malloc(sizeof(TLista));
        novo->info = i;
        novo->prox = NULL;
        TLista* p = li;
        if (p == NULL) { //se a lista estiver vazia
                li = novo;
        } else {
                while (p->prox != NULL) { //encontra o ultimo elemento
                        p = p->prox;
                }
                p->prox = novo;
        }
        return li;
}
```

Insere fim - Alternativa recursiva

```
TLista* insere_fim_recursivo (TLista* li, int i) {
        if (li == NULL || li->prox == NULL) {
                TLista *novo = (TLista *) malloc(sizeof(TLista));
                novo->info = i;
                novo->prox = NULL;
                if (li == NULL) {
                        li = novo;
                } else li->prox = novo;
        }
        else insere_fim_recursivo(li->prox, i);
        return li;
}
```

Cuidados ao utilizar funções recursivas

Tomar cuidado com loops infinitos ou muito longos.

Pode estourar a memória no meio da execução.

Prejudica legibilidade de código.

Exercícios

1 - Criar uma lista de inteiros que todo novo elemento é inserido de forma ordenada.

2 - Função para deletar um elemento da lista

3 - Função para alterar um elemento da lista

Pilhas e filas

Pilha

- Só movimenta o topo
- Inserção, remoção, consulta, tudo é feito no topo da pilha

Cria pilha

```
typedef struct pilha{
        TLista *topo;
} TPilha;
```

Inicialização

```
TPilha *inicializa() {
        TPilha *pilha = (TPilha *)malloc(sizeof(TPilha));
        pilha->topo = NULL;
        return pilha;
}
int main() {
TPilha *pilha = inicializa();
...
}
```

Push - Insere elemento no topo da pilha

```
void push(TPilha *pilha, int elem) {
        TLista *novo = (TLista*) malloc(sizeof(TLista));
        novo->info = elem;
        novo->prox = pilha->topo;
        pilha->topo = novo;
}
```

Pop - retira o elemento da pilha retornando o valor

```
int pop(TPilha *pilha) {
    if (pilha_vazia(pilha)) {
        exit(1);
    }
    else {
        TLista *aux = pilha->topo;
        int info = aux->info;
        pilha->topo = aux->prox;
        free(aux);
        return info;
    }
}
```

```
int pilha_vazia(TPilha *pilha) {
    if (pilha->topo == NULL)
        return 1; //pilha vazia
    else
        return 0; //pilha tem pelo menos 1 elemento
}
```

Peek - retorna o valor do topo sem remover

```
int peek(TPilha *pilha) {
    if (pilha_vazia(pilha))
        return NULL;
    else {
        return pilha->topo->info;
    }
}
```

Exercício:

Imprimir o conteúdo da pilha usando as funções push e pop.

Imprimir a pilha usando push e pop

```
void imprime_pilha(TPilha *pilha) {
    int x;
    printf("\nEstado atual da Pilha:\n");
    TPilha *aux = inicializa();
    while (!pilha_vazia(pilha)) {
        x = pop(pilha);
        printf("%d\n", x);
        push(aux, x);
    } while (!pilha_vazia(aux)) {
        push(pilha, pop(aux));
    }
    libera(aux);
    printf("\n");
}
```

Libera toda a memória da pilha

```
void libera(TPilha *p) {
    TLista *q = p->topo;
    TLista *r;
    while(q != NULL){
        r = q;
        q=q->prox;
        free(r);
    }
    free(p);
}
```

Fila

- Movimenta o início e o fim da lista.
- Inserção - Sempre no fim de fila
- Remoção - Sempre do início da fila
- Consulta - Início da fila

Declaração

```
#include "lista-encadeada.h"

typedef struct fila {
        TLista *inicio;
        TLista *fim;
} TFila;
```

Insere - No fim da fila

```
void insere(TFila *f, int elem){
        TLista *novo = (TLista *)malloc(sizeof(TLista));
        novo->info = elem;
        novo->prox = NULL; //inserção no fim da fila
        if (!fila_vazia(f)){
                f->fim->prox = novo;
        } else {
        f->inicio = novo;
}
        f->fim = novo; //elt. novo é o novo fim da fila
}
```

```
int fila_vazia(TFila *f){
        if (f->inicio == NULL) {
                return 1;
}
        else return 0;
}
```

Retira - Sempre do início da fila

```
int retira(TFila *f){
        if (fila_vazia(f)){
                exit(1);
        }
        int info = f->inicio->info;
        TLista *aux = f->inicio;
        f->inicio = f->inicio->prox;
        if (f->inicio == NULL) { // se a fila ficou vazia
                f->fim = NULL;
        }
        free(aux);
        return info;
}
```

Exercício

Faça a função que imprime o conteúdo da fila.