Capacitação em Linguagem C
Parte 2

Andrey Souto Maior
Giuseppe Portolese

Universidade Estadual de Maringá - Centro de Tecnologia
Departamento de Informática

22 de outubro de 2015

Sumário I

Struct

Uma estrutura (*struct*) é uma coleção de uma ou mais variáveis, possivelmente de tipos diferentes, colocadas juntas sob um único nome para manipulação conveniente.

Exemplo:

```
struct ponto {
    int x;
    int y;
};
```

```
struct {
    char id[10];
    int tam;
} x, y, z;
```

Pergunta: Qual a diferença entre as duas declarações?

Exemplo

```c
#include <stdio.h>

struct PONTO{
    int posicaoX;
    int posicaoY;
};

struct PONTO pontos[10];
```

Typedef

Typedef é um comando que permite criar novos tipos de dados. Por exemplo, a declaração:

```
typedef int Tamanho;
```

torna o nome Tamanho um sinônimo para int.

Exemplo:

```
typedef int *Vetor;
Vetor vet[10];
for (i = 0; i <10; i++) scanf("%d", &vet[i]);
```

Desafio

Faça um programa que receba o nome e a idade de 5 pessoas e depois os escreva na tela.

Desafio

```c
#include <stdio.h>

typedef struct{
    char nome[50];
    int idade;
}PESSOA;

PESSOA usuarios[5];
```

Desafio

```
int main(){
    int i;
    for(i=0; i<5; i++){
        printf("Digite o nome: ");
        scanf("%s",&usuarios[i].nome);
        printf("Digite a idade: ");
        scanf("%d",&usuarios[i].idade);

        printf("Nome: %s Idade: %d\n", usuarios[i].nome,
            usuarios[i].idade);
    }
    return 0;
}
```

Problema da ordenação

- **Entrada:** Uma sequência de n números $< a_1, a_2, a_3, ..., a_n >$.
- **Saída:** Uma permutação (reordenação) $< a'_1, a'_2, a'_3, ..., a'_n >$ da sequência de entrada, tal que $a'_1 \leq a'_2 \leq a'_3 \leq ... \leq a'_n$.

Exercício: Faça um algoritmo para ordenar um vetor de 10 elementos distintos.

Insertion-sort

A ordenação por inserção funciona da maneira como muitas pessoas ordenam cartas em um jogo de pôquer. Pegaremos uma carta de cada vez, vamos compará-la a cada uma das outras cartas, da direita para esquera, a fim de encontrar a posição correta.

Insertion-sort

A ordenação por inserção funciona da maneira como muitas pessoas ordenam cartas em um jogo de pôquer. Pegaremos uma carta de cada vez, vamos compará-la a cada uma das outras cartas, da direita para esquera, a fim de encontrar a posição correta.

```
void insertion(int vet[], int n){
    int i, j, k;
    for (j = 1; j < n; j++){
        int chave = vet[j];
        i = j-1;

        while((i >= 0) && (vet[i] > chave)){
            printf("%d %d\n", vet[i+1], vet[i]);
            vet[i+1] = vet[i];
            i--;
        }
        vet[i+1] = chave;
    }
}
```

Bubble-sort

Outro método de ordenação popular é o bubble-sort. Ele funciona permutando repetidamente elementos adjacentes que estão fora de ordem.

Bubble-sort

Outro método de ordenação popular é o bubble-sort. Ele funciona permutando repetidamente elementos adjacentes que estão fora de ordem.

```
void bubble(int vet[], int n){
    int i, j, aux;
    for (i = 0; i < n; i++){
        for (j = n-1; j > i; j--){
            if (vet[j] < vet[j-1]){
                aux = vet[j];
                vet[j] = vet[j-1];
                vet[j-1] = aux;
            }
        }
    }
}
```

Quick-sort

O quick-sort se baseia no paradigma de dividir e conquistar, já falado anteriormente.

- **Dividir:** O vetor $A[p..r]$ é particionado em dois subarranjos $A[p..q-1]$ e $A[q+1..r]$ tais que cada elemento de $A[p..q-1]$ é menor que ou igual a $A[q]$ que, por sua vez, é igual ou menor a cada elemento de $A[q+1..r]$. O índice q é calculado como parte desse procedimento de particionamento.
- **Conquistar:** Os dois $A[p..q-1]$ e $A[q+1..r]$ são ordenados por chamadas recursivas a quicksort.
- **Combinar:** Como os subarranjos são ordenados localmente, não é necessário nenhum trabalho para combiná-los: o arranjo $A[p..r]$ inteiro agora está ordenado.

Quick-sort

```
void quick(int vet[], int p, int r){
    int q;
    if (p < (r-1)){
        q = partition(vet, p, r);
        quick(vet, p, q);
        quick(vet, q+1, r);
    }
}
```

Quick-sort

```
int partition(int vet[], int p, int r){
    int x, i, j, aux;
    x = vet[r-1];
    i = p-1;
    for (j = p; j < r-1; j++){
        if (vet[j] <= x){
            i++;
            aux = vet[i];
            vet[i] = vet[j];
            vet[j] = aux;
        }
    }
    aux = vet[i+1];
    vet[i+1] = vet[j];
    vet[j] = aux;
    return i+1;
}
```

Merge-sort

O algoritmo de ordenação por intercalação (Mergesort) obedeçe o paradigma de dividir e conquistar.

- **Dividir:** Divide a sequência de N elementos a serem ordenados em duas sequências de N/2 elementos cada uma.
- **Conquistar:** Classfica as duas subsequências resurcisavemnte ustilizando a ordenação por intercalação.
- **Combinar:** Faz a intercalação das duas sequências ordenadas de modo a produzir a resposta ordenada.

Merge-sort

```
void mergeSort (int vet[], int p, int r){
if (p < r-1) {
        int q = (p + r)/2;
        mergeSort (p, q, v);
        mergeSort (q, r, v);
        merge (p, q, r, v);
    }
}
```

Merge-sort

```
void merge (int vet[], int p, int q, int r) {
    int i, j, k, *w;
    w = (int*) malloc ((r-p) * sizeof (int));
    i = p; j = q; k = 0;
    while (i < q && j < r) {
        if (atoi(vet[i].key) < atoi(vet[j].key)){
            w[k++] = v[i++];
        } else {
            w[k++] = v[j++];
        }
    }
    while (i < q) w[k++] = vet[i++];
    while (j < r) w[k++] = vet[j++];
    for (i = p; i < r; ++i) {
        vet[i] = w[i-p];
    }
    free (w);
}
```

Listas Simples e duplamente ligadas

Listas são estruturas de dados lineares e dinâmicas. Elas consistem em "Nós"que possuem dados (ou valores) e um ponteiro para o próximo Nó da lista ou para o vazio (Null)

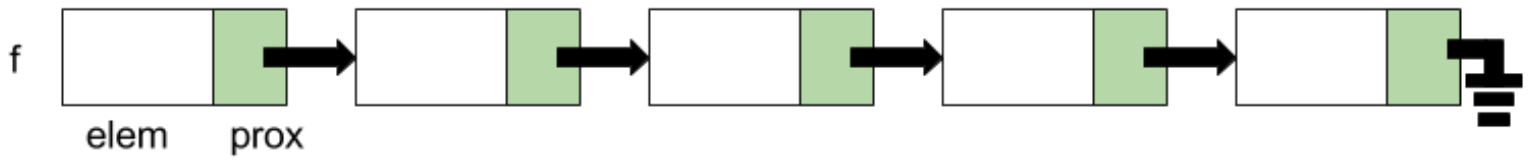

Figura 1: Lista Simples

Listas Simples e duplamente ligadas

```
typedef struct No_st{
    int numero;
    struct No_st *prox;
}No;
```

Listas Simples e duplamente ligadas

Também podemos fazer uma lista que seja Duplamente Ligada. Isso quer dizer que cada nó, alem de ter um ponteiro para o próximo elemento da lista tambem possui um ponteiro para o elemento anteiror.

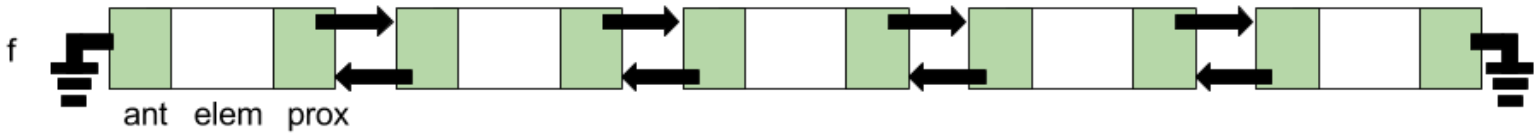

Figura 2: Lista Dupla

Listas Simples e duplamente ligadas

```
typedef struct lista_int{
    int numero;
    struct lista_int *seg;
    struct lista_int *ant;
}lista_dupla;
```

Criação

Para criar nossa fila não basta que só atribuamos valores para a variável. Precisamos inicializar o "Topo"de maneira correta, alocando o espaço necessário para nosso nó e dizendo ao computador que o próximo elemento do nosso topo será nulo.

```
No * cria_lista(){
    No * novo,*aux;
    novo = (No *) malloc( sizeof( No ));

    if(novo == NULL) exit(0);
    novo->prox = NULL;

    aux= novo;

    return (aux);
}
```

Criação

```
int main(void){
    No * raiz;

    raiz = cria_lista();
    return 0;
}
```

Inserção

```
No * inserirNoInicio(No * raiz, int numero){
    No * novo, *aux;
    aux = raiz;
    novo = (No *) malloc( sizeof(No) );
    if(novo == NULL) exit(0);
    novo->numero = numero;
    novo->prox = aux->prox;
    aux->prox = novo;
    return(aux);
}
```

Remoção

```
void removerNoInicio(No *raiz){
    No *aux;
    if(raiz == NULL)
        printf("\nA lista ja esta vazia");
    else{
        aux = raiz->prox;
        raiz->prox = aux->prox;
        free(aux);
    }
}
```

Pilha

Uma pilha é uma lista em que todas as inserções e remoções são feitas no topo (primeiro elemento). Temos que "Empilhar" nossos valores e depois para removê-los precisamos "Desempilhar".

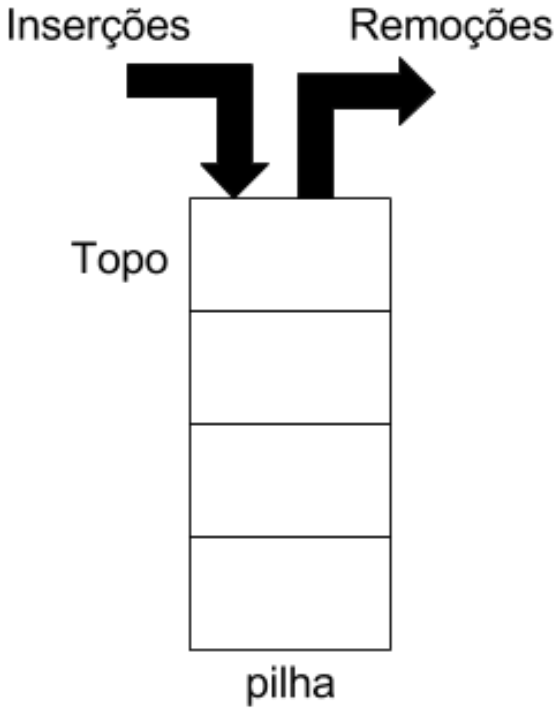

Figura 3: Pilha

Fila

Uma fila é uma lista em que fazemos todas as inserções na cauda (último elemento), enquanto que todas as remoções são feitas no topo. O mesmo funcionamento de filas que vemos no dia-a-dia em supermercados, bancos, etc.

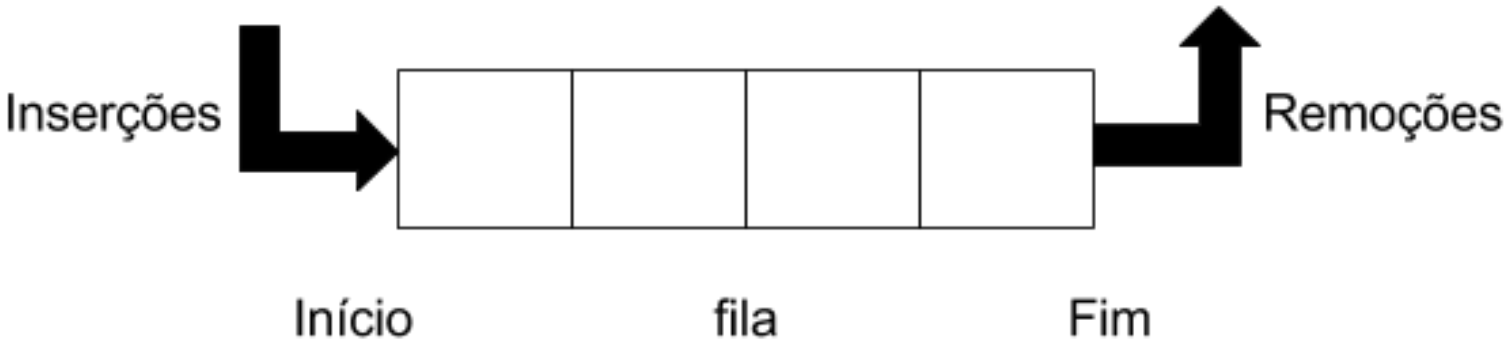

Figura 4: Fila

Pesquisa sequencial

Tambem chamada de Busca Linear, consiste em analisar elementos um por um até que se ache o elemento esperado.

Pesquisa sequencial

Tambem chamada de Busca Linear, consiste em analisar elementos um por um até que se ache o elemento esperado.

```
int procura(char vetor[], int tamanho, char
    elementoProcurado) {
    int i;
    for (i = 0; i < tamanho; i++) {
        if (vetor[i] == elementoProcurado) {
            return i;
        }
    }

    return -1;
 }
```

Pesquisa binária

Esta estratégia assume que o vetor em qual estamos buscando esteja ordenado. A busca binária separa o vetor em duas partes iguais sucessivamente e verifica se o valor do meio é maior, menor ou igual ao que estamos procurando e então repetimos a busca no vetor necessário.

Pesquisa binária

```
int PesquisaBinaria ( int vet[], int chave, int Tam){
    int inf = 0;
    int sup = Tam-1;
    int meio;
    while (inf <= sup){
        meio = (inf + sup)/2;
        if (chave == vet[meio])
            return meio;
        else if (chave < vet[meio])
            sup = meio-1;
        else
            inf = meio+1;
    }
    return -1;
}
```