# Estrutura de Dados

Objetivo da disciplina: aprender a criar algoritmos, listas, grafos, tipos de buscas e sort.

Linguagem utilizada: C

## **Semana 1**

**O que deve saber em C**, para entender Semana1:

* Tipos de dados (variáveis) (int, char,)
* #define, #include e typedef
* PrintF
* While, for (estruturas de repetição)
  + Semana 1, grupo de estudos C++ <https://github.com/dorathoto/UnivespC-1>
  + While -> <http://excript.com/cpp/estrutura-repeticao-while-cpp.html>
  + For -> <http://excript.com/cpp/estrutura-repeticao-for-cpp.html>
* Array
  + S
* Struct
  + <https://youtu.be/YeRebCWzO6E>
  + <https://youtu.be/MYe6vwhze9Q>
* Funções
  + <https://youtu.be/8Uvq3cNPTfY>
* Ponteiros
  + Part 1: <https://youtu.be/U3Y9d5Jz4LQ>
  + Part 2: <https://youtu.be/CrFq_L-UZ6g>
  + Part 3: <https://youtu.be/PGG6jr0uMyQ>
  + Me salva: <https://youtu.be/D5QvQmes198>

**Ponteiros:**

Ponteiros são muito úteis quando uma variável tem que ser acessada em diferentes partes de um programa.

Neste caso, o código pode ter vários ponteiros espalhados por diversas partes do programa, “apontando” para a variável que contém o dado desejado.

Caso este dado seja alterado, não há problema algum, pois todas as partes do programa tem um ponteiro que aponta para o endereço onde reside o dado atualizado.

Existem várias situações onde ponteiros são úteis, por exemplo:

* Alocação dinâmica de memória;
* Manipulação de arrays;
* Para retornar mais de um valor em uma função;
* Referência para listas, pilhas, árvores e grafos;

**Sintaxe de declaração de ponteiro**

tipo \*nome\_ponteiro;

Onde temos:

tipo: é o tipo de dado da variável cujo endereço o ponteiro armazena.

\*nome\_ponteiro: O nome da variável ponteiro.

O asterisco \* neste tipo de declaração determina que a variável será um ponteiro.

Exemplo de declaração de ponteiro:

int \*ptr;

Exemplo Lista ligada -> <https://pastebin.com/vxTjCi2B>

1. #include <stdio.h>
2. #define MAX 50
3. #define ERRO -1
4. #define true 1
5. #define false 0
7. typedef int bool;
9. typedef int TIPOCHAVE;
11. typedef struct{
12. TIPOCHAVE chave;
13. *// outros campos...*
14. } REGISTRO;
16. typedef struct {
17. REGISTRO A[MAX+1];
18. int nroElem;
19. } LISTA;
21. */\* Inicialização da lista sequencial (a lista já está criada e é apontada pelo endereço em l) \*/*
22. void inicializarLista(LISTA\* l){
23. l->nroElem = 0;
24. } */\* inicializarLista \*/*

27. */\* Exibição da lista sequencial \*/*
28. void exibirLista(LISTA\* l){
29. int i;
30. printf("Lista: **\"** ");
31. for (i=0; i < l->nroElem; i++)
32. printf("%i ", l->A[i].chave);
33. printf("**\"\n**");
34. } */\* exibirLista \*/*
36. */\* Retornar o tamanho da lista (numero de elementos "validos") \*/*
37. int tamanho(LISTA\* l) {
38. return l->nroElem;
39. } */\* tamanho \*/*
41. */\* Retornar o tamanho em bytes da lista. Neste caso, isto nao depende do numero*
42. *de elementos que estao sendo usados, pois a alocacao de memoria eh estatica.*
43. *A priori, nao precisariamos do ponteiro para a lista, vamos utiliza-lo apenas*
44. *porque teremos as mesmas funcoes para listas ligadas.*
45. *\*/*
46. int tamanhoEmBytes(LISTA\* l) {
47. return sizeof(LISTA);
48. } */\* tamanhoEmBytes \*/*
50. */\* Retornar a chave do primeiro elemento da lista sequencial (caso haja) e ERRO*
51. *caso a lista esteja vazia \*/*
52. TIPOCHAVE primeiroElem(LISTA\* l){
53. if(l->nroElem > 0) return l->A[0].chave;
54. else return ERRO; *// lista vazia*
55. } */\* primeiroElem \*/*
57. */\* Retornar a chave do ultimo elemento da lista sequencial (caso haja) e ERRO*
58. *caso a lista esteja vazia \*/*
59. TIPOCHAVE ultimoElem(LISTA\* l) {
60. if(l->nroElem > 0) return l->A[l->nroElem-1].chave;
61. else return ERRO; *// lista vazia*
62. } */\* ultimoElem \*/*
64. */\* Retornar a chave do elemento que está na posição n da LISTA. Lembre-se que as posicoes do*
65. *arranjo A vao de 0 a MAX-1  \*/*
66. TIPOCHAVE enesimoElem(LISTA\* l, int n) {
67. if( (n >= 0) && (n < l->nroElem)) return l->A[n].chave ;
68. else return ERRO;
69. } */\* enesimoElem \*/*
71. */\* Reinicializar a estrutura \*/*
72. void reinicializarLista(LISTA\* l) {
73. l->nroElem = 0;
74. } */\* reinicializarLista \*/*

77. */\* Busca sequencial em lista ordenada ou não SEM SENTINELA \*/*
78. int buscaSequencial(LISTA\* l, TIPOCHAVE ch) {
79. int i = 0;
80. while (i < l->nroElem){
81. if(ch == l->A[i].chave) return i; *// achou*
82. else i++;
83. }
84. return ERRO; *// não achou*
85. } */\* buscaSequencial \*/*

88. */\* Busca sequencial em lista COM SENTINELA (vetor criado com MAX+1 posições) \*/*
89. int buscaSentinela(LISTA\* l, TIPOCHAVE ch) {
90. int i = 0;
91. l->A[l->nroElem].chave = ch; *// sentinela*
92. while(l->A[i].chave != ch) i++;
93. if (i > l->nroElem -1) return ERRO; *// não achou*
94. else return i;
95. } */\* buscaSentinela \*/*
97. */\* Busca sequencial em lista COM SENTINELA (vetor criado com MAX+1 posições)*
98. *considerando o arranjo ordenado \*/*
99. int buscaSentinelaOrdenada(LISTA\* l, TIPOCHAVE ch) {
100. int i = 0;
101. l->A[l->nroElem].chave = ch; *// sentinela*
102. while(l->A[i].chave < ch) i++;
103. if (i == l->nroElem || l->A[i].chave != ch) return ERRO; *// não achou*
104. else return i;
105. } */\* buscaSentinela \*/*
107. */\* Busca binaria em lista ordenada \*/*
108. int buscaBinaria(LISTA\* l, TIPOCHAVE ch){
109. int esq, dir, meio;
110. esq = 0;
111. dir = l->nroElem-1;
112. while(esq <= dir) {
113. meio = ((esq + dir) / 2);
114. if(l->A[meio].chave == ch) return meio; *// achou*
115. else {
116. if(l->A[meio].chave < ch) esq = meio + 1;
117. else dir = meio - 1;
118. }
119. }
120. return ERRO;
121. } */\* buscaBinaria \*/*

124. */\* Exclusão do elemento cuja chave seja igual a ch \*/*
125. bool excluirElemLista(LISTA\* l, TIPOCHAVE ch) {
126. int pos, j;
127. pos = buscaSequencial(l,ch);
128. if(pos == ERRO) return **false**; *// não existe*
129. for(j = pos; j < l->nroElem-1; j++) l->A[j] = l->A[j+1];
130. l->nroElem--;
131. return **true**;
132. } */\* excluirElemLista \*/*

135. */\* Exclusão do elemento cuja chave seja igual a ch em lista ordenada\*/*
136. bool excluirElemListaOrd(LISTA\* l, TIPOCHAVE ch) {
137. int pos, j;
138. pos = buscaBinaria(l,ch);
139. if(pos == ERRO) return **false**; *// não existe*
140. for(j = pos; j < l->nroElem-1; j++) l->A[j] = l->A[j+1];
141. l->nroElem--;
142. return **true**;
143. } */\* excluirElemListaOrd \*/*

146. */\* Inserção em lista ordenada usando busca binária permitindo duplicação \*/*
147. bool inserirElemListaOrd(LISTA\* l, REGISTRO reg) {
148. if(l->nroElem >= MAX) return **false**; *// lista cheia*
149. int pos = l->nroElem;
150. while(pos > 0 && l->A[pos-1].chave > reg.chave) {
151. l->A[pos] = l->A[pos-1];
152. pos--;
153. }
154. l->A[pos] = reg;
155. l->nroElem++;
156. return **true**;
157. } */\* inserirElemListaOrd \*/*


161. */\* Inserção em lista ordenada usando busca binária sem duplicação \*/*
162. bool inserirElemListaOrdSemDup(LISTA\* l, REGISTRO reg) {
163. if(l->nroElem >= MAX) return **false**; *// lista cheia*
164. int pos;
165. pos = buscaBinaria(l,reg.chave);
166. if(pos != ERRO) return **false**; *// elemento já existe*
167. pos = l->nroElem-1;
168. while(pos>0 && l->A[pos].chave > reg.chave) {
169. l->A[pos+1] = l->A[pos];
170. pos--;
171. }
172. l->A[pos+1] = reg;
173. l->nroElem++;
174. return **true**;
175. } */\* inserirElemListaOrd \*/*


179. int main(){
180. LISTA lista;
181. inicializarLista(&lista);
182. exibirLista(&lista);
183. printf("Numero de elementos na lista: %i.**\n**",tamanho(&lista));
184. printf("Tamanho da lista (em bytes): %i.**\n**",tamanhoEmBytes(&lista));
185. REGISTRO reg;
186. reg.chave = 9;
187. inserirElemListaOrd(&lista,reg);
188. exibirLista(&lista);
189. reg.chave=3;
190. inserirElemListaOrd(&lista,reg);
191. reg.chave=4;
192. inserirElemListaOrd(&lista,reg);
193. reg.chave=1;
194. inserirElemListaOrd(&lista,reg);
195. reg.chave=12;
196. inserirElemListaOrd(&lista,reg);
197. exibirLista(&lista);
198. printf("Numero de elementos na lista: %i.**\n**",tamanho(&lista));
199. printf("Tamanho da lista (em bytes): %i.**\n**",tamanhoEmBytes(&lista));
200. printf("Chave 4 encontrada na posicao: %i do arranjo A.**\n**",buscaSequencial(&lista,4));
201. printf("Chave 4 encontrada na posicao: %i do arranjo A.**\n**",buscaBinaria(&lista,4));
202. printf("Chave 4 encontrada na posicao: %i do arranjo A.**\n**",buscaSentinela(&lista,4));
203. if (excluirElemLista(&lista,4)) printf("Exclusao bem sucedida: 4.**\n**");
204. if (excluirElemLista(&lista,8)) printf("Exclusao bem sucedida: 8.**\n**");
205. if (excluirElemLista(&lista,9)) printf("Exclusao bem sucedida: 9.**\n**");
206. exibirLista(&lista);
207. printf("Numero de elementos na lista: %i.**\n**",tamanho(&lista));
208. printf("Tamanho da lista (em bytes): %i.**\n**",tamanhoEmBytes(&lista));
209. reinicializarLista(&lista);
210. exibirLista(&lista);
211. printf("Numero de elementos na lista: %i.**\n**",tamanho(&lista));
212. printf("Tamanho da lista (em bytes): %i.**\n**",tamanhoEmBytes(&lista));
213. return 0;
214. }

**FAZER EXERCÍCIOS DE APOIO**

## **Semana 2**

O que aprender para Semana 2:

* Nenhuma sintaxe nova, apenas o conceito de pilha que é um vetor.
  + <https://www.youtube.com/watch?v=Symvpn9J3FM>
  + Plus: entender LIFO (Last in, first out - Último que entra, primeiro que sai)
  + Plus: Push e POP em listas
* Aprender conceito da busca binária

A busca binária ou pesquisa binária parte que sua lista (vetor) esteja ordenada então se divide ao meio, assim verifica se o numero procurado é maior que o número do meio, se for utilize a metade maior, caso contrário utilize a metade menor, e volte a dividir essa metade até que o número escolhido seja o do ‘meio’.

A busca binária é um tipo de busca não eficiente, é importante entender ela para que depois possa entender outros tipos de buscas e suas vantagens.

**FAZER EXERCÍCIOS DE APOIO**