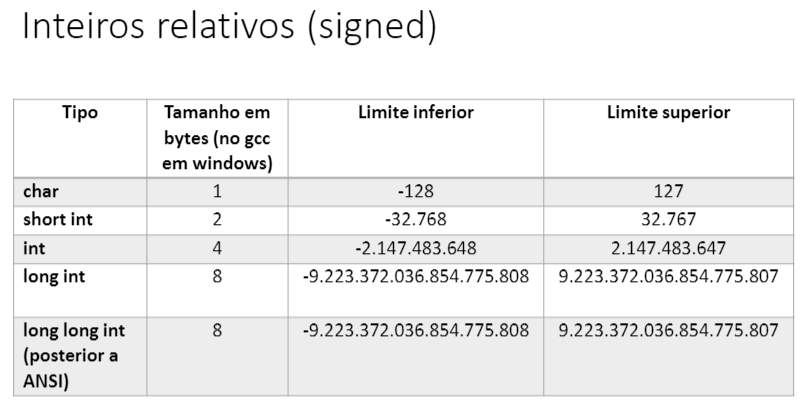
CONSULTA

#include <stdio.h> #include <string.h> #include <math.h>

 Uma imagem com mesa

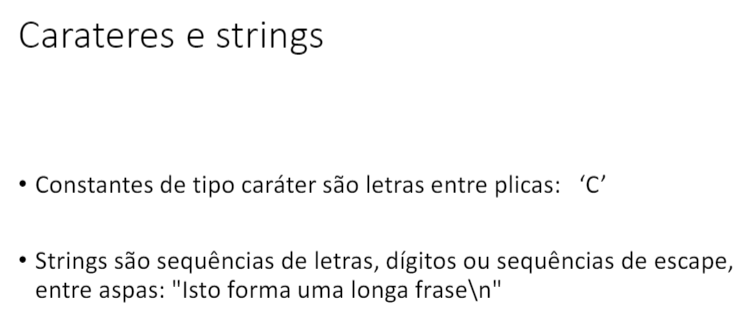
Descrição gerada automaticamente

Uma imagem com mesa

Descrição gerada automaticamente Uma imagem com texto, carta

Descrição gerada automaticamente

Uma imagem com texto

Descrição gerada automaticamente 

Uma imagem com texto

Descrição gerada automaticamente Uma imagem com texto

Descrição gerada automaticamente

Uma imagem com texto

Descrição gerada automaticamente Uma imagem com texto

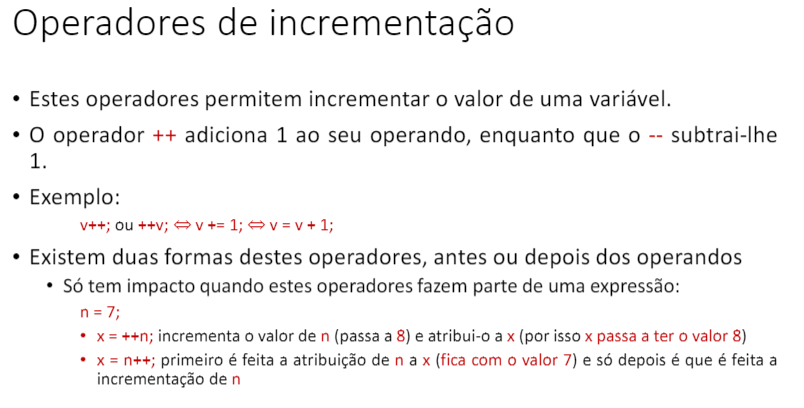
Descrição gerada automaticamente

Uma imagem com texto

Descrição gerada automaticamente

Uma imagem com diagrama

Descrição gerada automaticamente Uma imagem com calendário

Descrição gerada automaticamente

Uma imagem com texto

Descrição gerada automaticamente Uma imagem com texto

Descrição gerada automaticamente

Uma imagem com gráfico

Descrição gerada automaticamente Uma imagem com texto

Descrição gerada automaticamente

Uma imagem com texto

Descrição gerada automaticamente Uma imagem com texto

Descrição gerada automaticamente

Uma imagem com texto

Descrição gerada automaticamente Uma imagem com texto

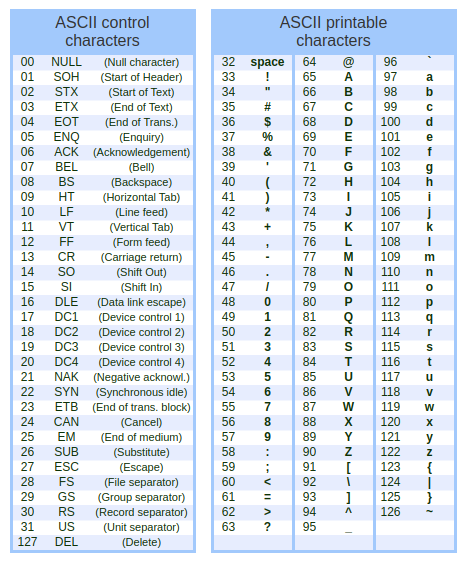
Descrição gerada automaticamente

Uma imagem com texto

Descrição gerada automaticamente

NOTAS

1. Printf(“…\n”) -> Não esquecer “\n”
2. Int x; -> Não esquecer “;” no final das linhas
3. Int main(void){… return 0;} -> Não esquecer “return 0;”
4. Função void -> não esquecer “return;”
5. Colocar a função declarada antes do “*int* main(*void*)”
6. Confirmar o tipo do return nas funções (*int*, *void*, …)
7. Confirmar parametros antes de “int main()” e na função criada
8. Scanf(“%s”, s) -> apenas lê até ao espaço ‘ ‘
9. Fgets(s, MAX, stdin) -> lê até ao enter



FUNÇÕES

|  |  |
| --- | --- |
| Char para Int:  *int* str\_int(*char* *s*[]){  *int* r = 0;      for(*int* i = 0; *s*[i] != '\0'; i++){          r = r \* 10 + *s*[i] - '0';      }      return r;  } | Permutações:  *int* permuts(*int* *n*, *int* *k*){  *int* num = 1;  *int* a = *n* - *k*;      for(*int* i = *n*; i > a; i--){          num \*= i;      }        return num;  } |
| Numeros Primos  *void* numerosPrimo(*int* n, *int* m){  *int* v = 0;      for(*int* i = n; i <= m; i++){          for(*int* j = 2; j < i; j++){              if(i % j == 0){                  v = 1;                  break;              }          }          if(v == 0){              printf("%d ", i);          }          v = 0;      }      printf("\n");  } | Lista dos primeiros Primos:  void primos(int x, int v[x]){      // Percorre numeros inteiros a começar em 2      for(int n\_atual = 2, tam\_v = 0; tam\_v < x; n\_atual++){          int br = 1;          for(int j = 0; j < tam\_v && v[j] \* v[j] <= n\_atual; j++){              if(n\_atual % v[j] == 0){                  br = 0;                  break;              }          }          if(br){              v[tam\_v] = n\_atual;              tam\_v++;          }      }  } |
| Inverter um vetor:  void inverte(int x, int v1[x], int v2[x]){      for(int i = 0; i < x; i++){          v2[x - i - 1] = v1[i];      }  } | Produto Matrizes:  void produto(int m, int n, int p, int a[m][n], int b[n][p], int prod[m][p]){      for(int i = 0; i < m; i++){          for(int k = 0; k < p; k++){              prod[i][k] = 0;              for(int j = 0; j < n; j++){                  prod[i][k] += a[i][j] \* b[j][k];              } }  } } |
| Ocorrencia numa String:  int ocor(char A[], char B[]){      int a, b;      int len\_B = strlen(B);      int len\_A\_B = strlen(A) - len\_B;      for(a = 0; a <= len\_A\_B ; a++){          for(b = 0; b < len\_B && A[a+b] == B[b]; b++){}          if(b == len\_B)              return a;      }      return -1;  } | Imprimir Tabela:  void tabela(int lin, int col, int tab[lin][col]){      for(int l = 0; l < lin; l++){          for(int c = 0; c < col; c++){              printf("%8d ", tab[l][c]);          }          printf("\n");      }  } |
| 2 vetores para 1 ordenado:  void vetores\_2\_para\_1\_ordenado(int n, int v1[n], int m, int v2[m], int o, int v3[o]){      int v\_1 = 0;      int v\_2 = 0;      for(int i = 0; i < n + m; i++){          if(v\_1 < n && v\_2 < m){              if(v1[v\_1] > v2[v\_2]){                  v3[i] = v2[v\_2];                  v\_2 += 1;              } else if(v1[v\_1] < v2[v\_2]){                  v3[i] = v1[v\_1];                  v\_1 += 1;              } else{                  v3[i] = v1[v\_1];                  v\_1 += 1;              }          } else if(v\_1 == n && v\_2 < m){              v3[i] = v2[v\_2];              v\_2 += 1;          } else if(v\_2 == m && v\_1 < n){              v3[i] = v1[v\_1];              v\_1 += 1;          }      }      return;  } | Ordenar vetor:  void ordenar\_datas(int n, Data data[n]){      int mudou = 1;  // se o vetor ja estiver ordenado, nao continua o ciclo for -> pára      for(int j  = 0; mudou && j < n - 1; j++){          mudou = 0;          for(int i = 0; i < n - 1; i++){              printf("%d %d\n", j, i);        // para ver que depois de estar ordenado - pára              if(compara\_datas(data[i], data[i + 1]) > 0){                  mudou = 1;                  Data atual = data[i];                  data[i] = data[i + 1];                  data[i + 1] = atual;              }          }      }  } |
| String para INT:  int main(void){      char s[MAX] = {'1', '9', '5', '7', '2', '3', '\0'};      printf("%d \n", str\_int(s));      return 0;  }  int str\_int(char s[]){      int r = 0;      for(int i = 0; s[i] != '\0'; i++){          r = r\*10 + s[i] - '0';      }  return r;  } | Numero de palavras em String:  int n\_pal(char s[]){      int pal = 1;      for(int i = 0; s[i] != '\0'; i++){          if(s[i] == ' ')              pal++;      }      return pal;  } |
| Minusculas para Maiusculas:  int main(void){      char frase[MAX] = "Frase com min e MAIUscuLas";      printf("antes: %s\n", frase);      min\_para\_mai(frase);      printf("depois: %s\n", frase);  }  void min\_para\_mai(char s[]){      for(int i = 0; s[i] != '\0'; i++){          if(s[i] >= 'a' && s[i] <= 'z'){              s[i] = s[i] - 'a' + 'A';          }      }      return;  } | Codificar e descodificar uma String:  void cod(char s[], char a[]){      for(int i = 0; s[i] != ‘\0’; i++){          int ascii = s[i];          if(ascii >= ‘a’ && ascii <= ‘z’){              ascii += ‘A’ – ‘a’;          } if(ascii >= ‘A’ && ascii <= ‘Z’){              s[i] = a[ascii – ‘A’];          }      } return; }  void inverte\_chave(int n, char chave[], char chave2[]){      for (int i = 0; i < n; i++){          int pos = chave[i] – ‘A’; // posiçao da letra chave[i] na nova chave (chave2[])          chave2[pos] = ‘A’ + i;      } return; } |
| Notação Cinetífica:  void not\_cien(char num[], char not[]){      if(num[1] != '0'){          not[0] = num[0];          not[1] = num[1];          not[2] = '.';          int p = -1;          int expoente;          for(int i = 3; num[i] != '\0'; i++){              if(p < 0){                  not[i] = num[i-1];                  expoente = i - 2;              } else{                  not[i] = num[i];              }          }            not[7] = 'E';          not[8] = '+';          not[9] = expoente + '0';      } else{          int expoente = -1;          for(int i = 0; not[i] != '\0'; i++){              not[i] = 0 + '0';          }            not[0] = num[0];          not[2] = '.';          for(int i = 3; num[i] != '\0'; i++){              if(expoente < 0){                  if(num[i] != '0'){                      not[1] = num[i];                      expoente = i - 2;                  }              } else{                  not[i - expoente] = num[i];              }          }          not[7] = 'E';          not[8] = '-';          not[9] = expoente + '0';      }  } |  |

Uma imagem com texto

Descrição gerada automaticamenteUma imagem com texto

Descrição gerada automaticamente

Uma imagem com texto

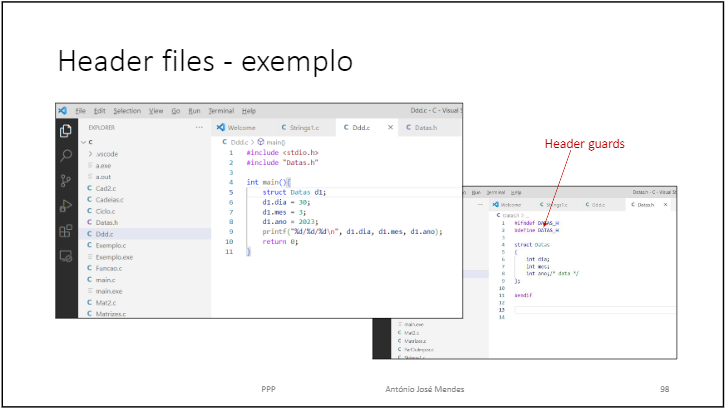
Descrição gerada automaticamenteUma imagem com texto

Descrição gerada automaticamente

Uma imagem com texto

Descrição gerada automaticamenteUma imagem com texto

Descrição gerada automaticamente

Uma imagem com texto

Descrição gerada automaticamente

Uma imagem com texto

Descrição gerada automaticamenteUma imagem com texto

Descrição gerada automaticamente

Uma imagem com texto

Descrição gerada automaticamenteUma imagem com texto

Descrição gerada automaticamente

Uma imagem com texto

Descrição gerada automaticamente

FUNÇÕES

|  |  |
| --- | --- |
| int compara(DATA d1, DATA d2){  // devolve 1 de d1 for maior e -1 se d2 for maior!  // devolve 0 se forem iguais  if(d1.ano > d2.ano)  return 1;  if(d1.ano < d2.ano)  return -1;  if(d1.mes != d2.mes)  return (d1.mes > d2.mes) - (d1.mes < d2.mes);  else{  return (d1.dia > d2.dia) - (d1.dia < d2.dia);  }  } | void extremos(int \* p\_M, int \* p\_m, int n, int \* v){  \* p\_M = \*p\_m = \*v;  for(int i = 1; i < n; i++){  if(\*(v + i) > \*p\_M){  \*p\_M = \*(v + i);  }  else if(\*(v + i) < \*p\_m){  \*p\_m = \*(v + i);  }  printf("Atual : %d\n", \*(v + i));  printf("Menor : %d\n\n", \*p\_m);  }  } |
| DATA dif(DATA d1, DATA d2){  DATA diferenca;  diferenca.dia = d1.dia - d2.dia;  diferenca.mes = d1.mes - d2.mes;  diferenca.ano = d1.ano - d2.ano;  return diferenca;  } | int sec(DATA d){  int s = ((d.ano - 1) / 100) + 1;  return s;  } |
| char \*maior\_pal(char \*s){  char delim[] = " \n\t,.?!:;\"'";  char \*p;  char \*m = NULL;  int len\_maior = 0;  p = strtok(s, delim);  while(p != NULL){ //ou for(p = strtok(s, delim); p != NULL; p = strtok(NULL, delim)){}  int t = strlen(p);  if(t > len\_maior){  len\_maior = t;  m = p;  }  p = strtok(NULL, delim);  }  return m;  } | int proc\_wally(char \* string){  char w[] = "Wally";  for(int i = 0; i < 5; i++){  printf("%d ", w[i]);  } printf("\n");  for(int i = 0; \*(string + i) != '\0'; i++){  printf("o = %d -> %d\n", i, \*(string + i));  if(\*(string + i) == w[0]){  for(int j = i; \*(string + i) != '\0' && \*(string + j) == w[j - i]; j++){  printf("Aqui\n");  if(j == 5){  printf("Acabei!\n");  return i;  }  }  }  }  return -1;  } |
| Construir Matriz:  void const\_mat(int n, int \* v){  for(int i = 0; i < n; i++){  for(int j = 0; j < n; j++){  printf("A[%d][%d] = ", i, j);  scanf("%d", (v + n\*i + j));  }  }  } | Maior elemento nos ultimos k elementos  int maior(int k, int n, int \* v){  int max = \*(v + n - k);  for(int i = n - k; i < n; i++){  printf("Numero atual = %d\n\n", \*(v + i));  if(\*(v + i) > max){  max = \*(v + i);  }  }  return max;  } |

Estruturas de Dados

|  |  |  |
| --- | --- | --- |
| main.c | fila.c | fila.h |
| #include <stdio.h>  #include <stdlib.h>  #include "lista.h"  int main(void){…} | #include <stdlib.h>  #include <stdio.h>  #include "lista.h"  \*Funções\* | #ifndef LISTA\_H  #define LISTA\_H  typedef struct noL{…}noLista;  \*Declaração Funções\*  #endif |

Uma imagem com diagrama

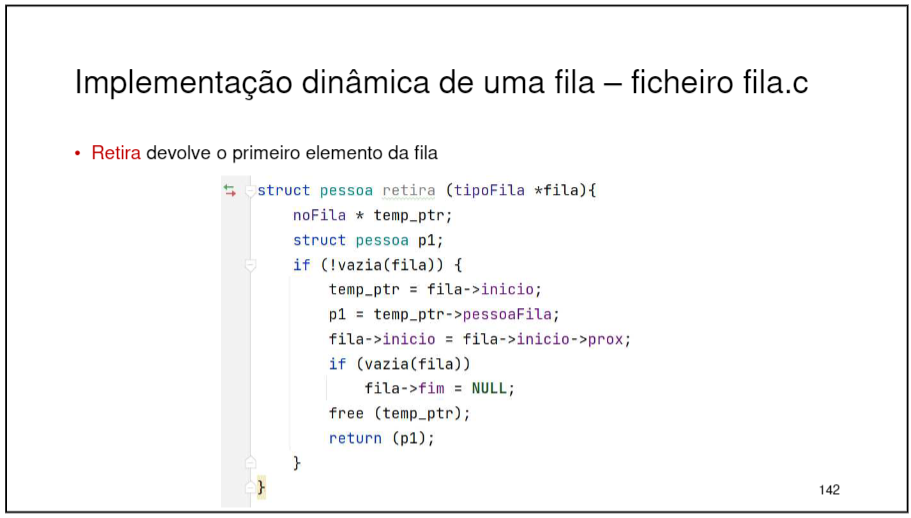
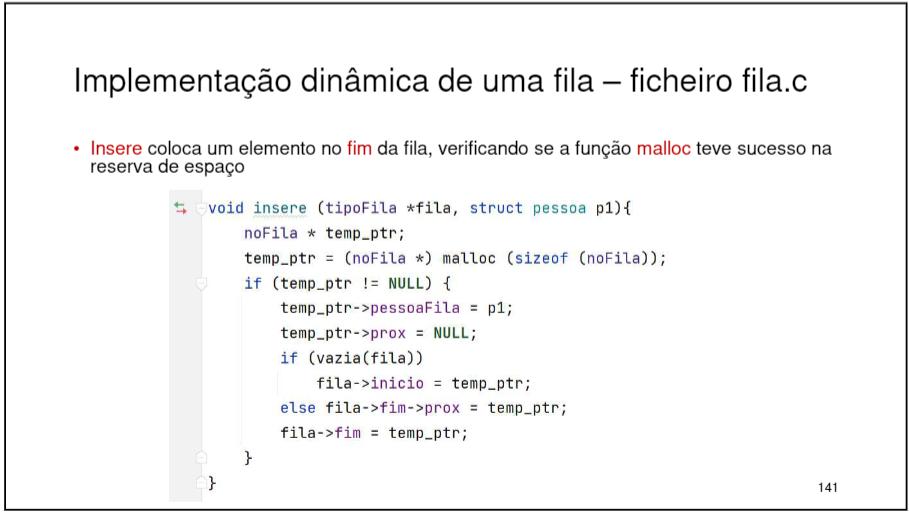
Descrição gerada automaticamenteUma imagem com texto

Descrição gerada automaticamente

Uma imagem com texto

Descrição gerada automaticamenteUma imagem com texto

Descrição gerada automaticamente



Notas

1. strlen() – unsigned long 🡪 %ul
2. 
3. scanf(“%d”, endereço);
   1. int \* v;
   2. scanf("%d", (v + n\*i + j)); // v é do tipo int \*
4. printf(“algo %d”, \*v); // \*v é do tipo int

Trabalhar com strings:

1. strcpy(\*dest, \*origem);
2. char \* s = strtok(\*s, \*delim);

Funções -> FILA

|  |  |
| --- | --- |
| void mostra(tipoFila \*fila){  noFila \*ptr;  for(ptr = fila -> inicio; ptr != NULL; ptr = ptr -> prox){  printf("%f\n", ptr -> x);  }  } |  |
| Passar de vetor a fila (com o nº de ocorrencias)  void f(int n, int v[n], Fila \* f1){  int max = v[0];  for(int i = 0; i < n; i++){  if(v[i] > max) max = v[i];  }  for(int i = 1; i < max + 1; i++){  int ocor = 0;  int b = -1;  for(int j = 0; j < n; j++){  if(i == v[j]){  ocor += 1;  b = 1;  }  }  if(b > 0){  insere(f1, i, ocor);  }  }  } | Abreviar (Artur Jose Costa -> Artur J. Costa)  char \* f(char nome[]){  char \* n = malloc(sizeof(char) \* 51);  int len = strlen(nome);  int ult\_esp;  for(int i = 0; i < len; i++){  if(nome[i] == ' ')  ult\_esp = i;  }  int i;  for(i = 0; nome[i] != ' '; i++){  \*(n + i) = nome[i];  }  \*(n + i) = ' ';  int j = i;  int b = 0;  for(i = j + 1; i < ult\_esp + 1; i++){  if(nome[i] != ' ' && b == 0){  \*(n + j + 1) = nome[i];  b = 1;  } else if(nome[i] == ' '){  \*(n + j + 2) = '.';  \*(n + j + 3) = ' ';  j += 3;  b = 0;  }  }  for(i = ult\_esp + 1; i < len; i++){  \*(n + j + i - ult\_esp) = nome[i];  }  return n;  } |
|  |  |
|  |  |
| STRINGS  void insere(tipoFila \*fila, char \* str){  noFila \* temp\_ptr;  temp\_ptr = (noFila \*) malloc (sizeof(noFila));  if(temp\_ptr != NULL){  strcpy(temp\_ptr -> s, str);  temp\_ptr -> prox = NULL;  if(vazia(fila))  fila -> inicio = temp\_ptr;  else fila -> fim -> prox = temp\_ptr;  fila -> fim = temp\_ptr;  }  } | STRINGS  int retira(tipoFila \*fila, char \*str){ /\* retira o primeiro elemento \*/  noFila \*ptr;  if(!vazia(fila)){  ptr = fila -> inicio;  strcpy(str, ptr -> s);  fila -> inicio = ptr -> prox;  if(vazia(fila)){  fila -> fim = NULL;  }  free(ptr);  return 0;  }  return -1;  } |

PILHAS

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente Uma imagem com texto, captura de ecrã, Tipo de letra

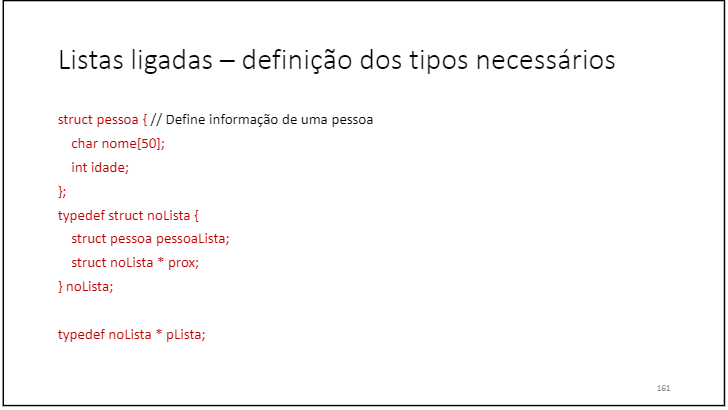
Descrição gerada automaticamente

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

LISTAS LIGADAS

 Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

Uma imagem com texto, captura de ecrã, Tipo de letra, recibo

Descrição gerada automaticamente Uma imagem com texto, captura de ecrã, Tipo de letra, número

Descrição gerada automaticamente

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente Uma imagem com texto, captura de ecrã, Tipo de letra, diagrama

Descrição gerada automaticamente

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

FICHEIROS

Uma imagem com texto, captura de ecrã, Tipo de letra, documento

Descrição gerada automaticamente Uma imagem com texto, captura de ecrã, Tipo de letra, número

Descrição gerada automaticamente

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

Uma imagem com texto, captura de ecrã, Tipo de letra, número

Descrição gerada automaticamente Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

Uma imagem com texto, captura de ecrã, Tipo de letra, número

Descrição gerada automaticamente Uma imagem com texto, captura de ecrã, Tipo de letra, número

Descrição gerada automaticamente

Uma imagem com texto, captura de ecrã, Tipo de letra, número

Descrição gerada automaticamente Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

Uma imagem com texto, captura de ecrã, Tipo de letra, número

Descrição gerada automaticamente Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

Uma imagem com texto, captura de ecrã, Tipo de letra, número

Descrição gerada automaticamente

Exercicios:

|  |  |
| --- | --- |
| ORDEM INVERSA - PILHA  #include <stdio.h>  #include "pilha.h"  int main(void){  int n, c, err;  noPilha \*p1;  // Criar a pilha  cria(&p1);  printf("Introduz os valores da tua pilha\n(Enter para introduzir) (ctrl + d para acabar):\n");  // Pedir numeros ao utilizador  c = scanf("%d", &n);  while(c == 1){  err = push(&p1, n);  if(err){  destroi(&p1);  return err;  }  c = scanf("%d", &n);  }  // Mostrar por ordem inversa  printf("\nA tua pilha, por ordem inversa, é:\n");  while(!vazia(&p1)){  err = pop(&p1, &n);  printf("%d\n", n);  }  return 0;  } | LISTAS LIGADAS – ORDEM CRESCENTE  #include <stdio.h>  #include <stdlib.h>  #include "lista.h"  int main(void){      int n, err;      double x;      noLista \*pl1;      /\* inicializar lista \*/      pl1 = cria();      if(pl1 == NULL){          perror("Erro de criação da lista\n");          return -1;      }      /\* Pedir dados ao utilizador \*/      printf("Quantos numeros pretende inserir?\n");      scanf("%d", &n);      for(int i = 0; i < n; i++){          printf("[%d]: ", i + 1);          scanf("%lf", &x);          err = insere(pl1,  x);          if(err)              perror("Erro de inserção\n");      }      printf("\n\tOrdenado\n\n");      imprime(pl1);      destroi(pl1);  } |

Listas Ligadas – imprimir por nome e por número

|  |  |
| --- | --- |
| #ifndef LISTA\_H  #define LISTA\_H  #define MAX\_NOME 80  typedef struct{  int num;  char nome[MAX\_NOME];  int dia, mes, ano; // Também podia ser uma struct Data...  int pontuacao;  } Atleta;  typedef struct noL{  Atleta dados;  struct noL \*prox; // Ordenaçao por numero do atleta  } noLista;  typedef struct noLNome{  Atleta \* dadosp;  struct noLNome \*prox; // Ordenaçao pelo nome do atleta, atraves de ponteiros  } noListaNome; // Desta forma não ocupa duas vezes a memória  typedef struct{ // Este cabeçalho aponta para duas listas diferentes  noLista \*prox; // Ordenação pelo numero do atleta  noListaNome \* prox\_nome; // Ordenaçao pelo nome do atleta  int tamanho;  } ListaAtletas;  ListaAtletas \*cria(void);  int vazia(ListaAtletas \* lista);  void destroi(ListaAtletas \* lista);  void procura\_num(ListaAtletas \*lista, int chave, noLista \*\*ant, noLista \*\*atual);  void procura\_nome(ListaAtletas \*lista, char \* chave, noListaNome \*\*ant, noListaNome \*\*atual);  //int elimina(noLista \*lista, double chave);  int insere(ListaAtletas \*lista, Atleta \*atletap);  noLista \*pesquisa(ListaAtletas \*lista, int chave);  void imprime\_por\_num(ListaAtletas \*lista);  void imprime\_por\_nome(ListaAtletas \*lista);  #endif | int insere(ListaAtletas \*lista, Atleta \*atletap){  noLista \*no, \*ant, \*atual;  noListaNome \*no\_Nome, \*ant\_Nome, \*atual\_Nome;    procura\_num(lista, atletap -> num, &ant, &atual);  if(atual != NULL){  }  no = (noLista \*) malloc (sizeof(noLista));  no\_Nome = (noListaNome \*) malloc(sizeof(noListaNome));  if(no != NULL && no\_Nome != NULL){  no -> dados = \*atletap;  // Insere na lista principal  if(ant == NULL){  no -> prox = lista -> prox;  lista -> prox = no;  } else{  no -> prox = ant -> prox;  ant -> prox = no;  }  // Insere na lista dos nomes  no\_Nome -> dadosp = &(no -> dados); // mesmo que '... = atletap'  procura\_nome(lista, atletap -> nome, &ant\_Nome, &atual\_Nome);  if(ant\_Nome == NULL){  no\_Nome -> prox = lista -> prox\_nome;  lista -> prox\_nome = no\_Nome;  } else{  no\_Nome -> prox = ant\_Nome -> prox;  ant\_Nome -> prox = no\_Nome;  }  return 0;  }  free(no);  free(no\_Nome);  perror("Erro de alocação de memória!");  return -1; // Erro de alocação de memória  } |

Ficheiros

|  |  |
| --- | --- |
| Linha mais longa – 14.1.1  #include <stdio.h>  #include <string.h>  #define LINE\_MAX 11  int main(void){  int n\_pal;  printf("Número de palavras a introduzir: ");  scanf("%d", &n\_pal);  FILE \*f, \*f2;  char pal[MAX];  f = fopen("palavras.txt", "w");  f2 = fopen("palin.txt", "w");  char maior\_pal[MAX] = {' '};  for(int i = 0; i < n\_pal; i++){  scanf("%s", pal);  fputs(pal, f);  // Se a nova palavra for maior que a anterior maior, atualizamos a maior palavra  if(strlen(maior\_pal) < strlen(pal))  strcpy(maior\_pal, pal);  if(palindromero(pal) == 0)  fprintf(f2, "%s\n", pal);  fputs("\n", f);  }  printf("A maior palavra é: %s\n", maior\_pal);    fclose(f);  fclose(f2);  } | Palavra mais longa, por no ficheiro – 14.2.1  #include <stdio.h>  #include <string.h>  #define MAX 15  int main(void){  int n\_pal;  printf("Número de palavras a introduzir: ");  scanf("%d", &n\_pal);  FILE \*f;  char pal[MAX];  f = fopen("palavras.txt", "w");  char maior\_pal[MAX] = {' '};  for(int i = 0; i < n\_pal; i++){  scanf("%s", pal);  fputs(pal, f);  // Se a nova palavra for maior que a anterior maior, atualizamos a maior palavra  if(strlen(maior\_pal) < strlen(pal))  strcpy(maior\_pal, pal);  fputs("\n", f);  }  fclose(f);  printf("A maior palavra é: %s\n", maior\_pal);  } |
| Palavras palindromas – igual de tras para a frente – 14.1.1  int palindromero(char pal[MAX]){  int len = strlen(pal);  for(int i = 0; i < len/2; i++){  if(pal[i] != pal[len - 1 - i])  return -1;  }  return 0;  } |  |
| Main.c – Ordenar alfabeticamente ficheiro binario com LISTA  #include <stdio.h>  #include "lista.h"  #define NUM\_ALUNOS 5  int main(void){  Pessoa alunos[NUM\_ALUNOS] = {  {"João", 2004},  {"Pedro", 2003},  {"Alexandra", 2004},  {"Thales", 2002},  {"Carlos", 1968}};  noLista \* lista = cria();  for(int i = 0; i < NUM\_ALUNOS; i++)  insere(lista, alunos + i); // ou insere(lista, &alunos[i]);  imprime(lista);  FILE \*fp = fopen("alunos.txt", "wb");  if(fp != NULL){  escreve(lista, fp);  fclose(fp);  }  destroi(lista);  return 0;  } | Lista.c – Strings e Binario  void procura(noLista \*lista, char \* chave, noLista \*\*ant, noLista \*\*atual){  \*ant = lista;  \*atual = lista -> prox;  while((\*atual) != NULL && strncmp((\*atual) -> pessoa.nome, chave, TAM\_NOME) < 0){  \*ant = \*atual;  \*atual = (\*atual) -> prox;  }  if((\*atual) != NULL && strncmp((\*atual) -> pessoa.nome, chave, TAM\_NOME) != 0)  \*atual = NULL; /\* elemento nao encontrado \*/  }  int insere (noLista \*lista, Pessoa \*pessoa){  noLista \*no, \*ant, \*inutil;  no = (noLista \*) malloc (sizeof(noLista));  if(no != NULL){  strcpy(no -> pessoa.nome, pessoa->nome);  no -> pessoa.ano = pessoa-> ano;  procura(lista, pessoa->nome, &ant, &inutil);  no -> prox = ant -> prox;  ant -> prox = no;  return 0;  }  return -1; // Erro de alocação de memória  }  void escreve(noLista \* lista, FILE \*fp){  noLista \*aux = lista -> prox; /\*Salta o header\*/  while(aux){  fwrite(&aux->pessoa.nome, sizeof(char), TAM\_NOME, fp);  fwrite(&aux->pessoa.ano, sizeof(int), 1, fp);  aux = aux -> prox;  }  } |

|  |  |
| --- | --- |
| Ler ficheiro binario e colocar em fila | Imprimir a maior nota e disciplina de todos os alunos |
| Um ficheiro binário contém apenas números inteiros, um por linha. O primeiro número indica  quantos números além dele existem no ficheiro. Escreva uma função e apenas uma que leia  os números do ficheiro e determine se seguem o padrão de estarem espelhados na segunda  metade do ficheiro em relação à primeira: | Dada uma lista de Pessoas, onde cada pessoa é representada com um identificador (idPessoa)  e uma lista de Items (itPessoa) cujos elementos também tem um identificador (idItem) se quer  determinar a interseção dos itens (elementos comuns) entre as pessoas da lista. Assuma que não  existem elementos repetidos dentro de uma lista de itens e que os números de elementos das listas  são iguais. As estruturas de dados e funções implementados neste programa são os seguintes: |