Algoritmos e Estruturas de Dados I

Prof. Lúcio Mauro Pereira

28/10/2024

Desafios propostos

Tente

Construa uma função que receba uma *string* e um arranjo de inteiros de tamanho igual a cinco. A função deverá preencher o vetor de inteiros com o número da vogal correspondente, considerando a seguinte sequência: ‘a’ na posição 0, ‘e’ na posição 1 e assim sucessivamente.

Tente

Construa uma função que receba três *strings.* A função deverá concatenar as duas primeiras *strings* na terceira.

Em discussão: vetores de estruturas e arquivos

Desafio em arranjos: tamanho

Proposta:

Tamanho **físico** em constante global

Tamanho **lógico** em variável global, inicializada com zero

Identificadores globais

```
const int MAX_STR =   50;
const int         MAX = 100;
          int         TAM =     0;
```

Estrutura exemplificada no laboratório

```
const int MAX_STR =   50;
...
typedef struct{
                    char nome[MAX_STR];
                    int    idade;
                } Pessoa;
```

Criando um vetor de pessoas com MAX elementos

```
#include <locale.h>
const int MAX_STR =   50;
const int        MAX = 100;
        int      TAM =    0;
typedef struct{
                    char nome[MAX_STR];
                    int   idade;
                  } Pessoa;
int main()
{
   setlocale(LC_ALL, "");
   abertura();



   return 0;
}
```

Criando um vetor de pessoas com MAX elementos

```
#include <locale.h>
const int MAX_STR =   50;
const int        MAX = 100;
        int        TAM =    0;
typedef struct{
                        char nome[MAX_STR];
                        int   idade;
                    } Pessoa;
int main()
{
   setlocale(LC_ALL, "");
   abertura();
   Pessoa PESSOAS[MAX];

   return 0;
}
```

Cadastrando uma pessoa

```
void cadastrePessoa(Pessoa PESSOAS[]){

    fflush(stdin); // Linux: __purge(stdin)
    printf("\nNome: ");
    fgets(PESSOAS[TAM].nome, MAX_STR, stdin);




}
```

Cadastrando uma pessoa

```
void cadastrePessoa(Pessoa PESSOAS[]){

    fflush(stdin); // Linux: __purge(stdin)
    printf("\nNome: ");
    fgets(PESSOAS[TAM].nome, MAX_STR, stdin);

    printf("\nIdade: ");
    scanf("%i", &PESSOAS[TAM].idade);


}
```

Cadastrando uma pessoa

```
void cadastrePessoa(Pessoa PESSOAS[]){

    fflush(stdin); // Linux: __purge(stdin)
    printf("\nNome: ");
    fgets(PESSOAS[TAM].nome, MAX_STR, stdin);

    printf("\nIdade: ");
    scanf("%i", &PESSOAS[TAM].idade);

    TAM++;
}
```

Escrevendo uma pessoa

```
void escrevePessoa(Pessoa PESSOAS[], int i){

    printf("\nNome: %s“, PESSOAS[i].nome);

    printf("\nIdade: %i“,   PESSOAS[i].idade);

}
```

Listando as pessoas

```
void listaPessoas(Pessoa PESSOAS[]){

    for(int i=0; i<TAM; i++){
        escrevePessoa(PESSOAS, i);
    }

}
```

Um menu de opções

```
int menu() {
    int OPCAO;


        printf("\n\nMenu de opções");
        printf("\n\t0 - SAIR");
        printf("\n\t1 - Cadastrar Pessoa");
        printf("\n\t2 - Listar pessoas");
        printf("\nSua Opção: ");
        scanf("%i", &OPCAO);



return OPCAO;
}
```

Um menu de opções

```
int menu() {
    int OPCAO;
    bool ERRO;

        printf("\n\nMenu de opções");
        printf("\n\t0 - SAIR");
        printf("\n\t1 - Cadastrar Pessoa");
        printf("\n\t2 - Listar pessoas");
        printf("\nSua Opção: ");
        scanf("%i", &OPCAO);
        ERRO = OPCAO<0 || OPCAO >2;


    return OPCAO;
}
```

Um menu de opções

```
int menu() {
    int OPCAO;
    bool ERRO;

        printf("\n\nMenu de opções");
        printf("\n\t0 - SAIR");
        printf("\n\t1 - Cadastrar Pessoa");
        printf("\n\t2 - Listar pessoas");
        printf("\nSua Opção: ");
        scanf("%i", &OPCAO);
        ERRO = OPCAO<0 || OPCAO >2;
        if(ERRO)printf("\nResposta inválida");

    return OPCAO;
}
```

Um menu de opções

```
int menu() {
    int OPCAO;
    bool ERRO;
    do{
        printf("\n\nMenu de opções");
        printf("\n\t0 - SAIR");
        printf("\n\t1 - Cadastrar Pessoa");
        printf("\n\t2 - Listar pessoas");
        printf("\nSua Opção: ");
        scanf("%i", &OPCAO);
        ERRO = OPCAO<0 || OPCAO >2;
        if(ERRO)printf("\nResposta inválida");
    }while(ERRO);
    return OPCAO;
}
```

Gerenciando tamanho lógico em arquivo

```
void abreArquivoTamanho() {
    FILE* arqTamanho = fopen("tamanho.dat","rb");




     fclose(arqTamanho);
}
```

Gerenciando tamanho lógico em arquivo

```
void abreArquivoTamanho() {
    FILE* arqTamanho = fopen("tamanho.dat","rb");

    if(arqTamanho == NULL){
        arqTamanho = fopen("tamanho.dat","wb");
        TAM=0;
        fprintf(arqTamanho, "%i", TAM);
    }
    else {

    }
    fclose(arqTamanho);
}
```

Gerenciando tamanho lógico em arquivo

```
void abreArquivoTamanho() {
    FILE* arqTamanho = fopen("tamanho.dat","rb");

    if(arqTamanho == NULL){
        arqTamanho = fopen("tamanho.dat","wb");
        TAM=0;
        fprintf(arqTamanho, "%i", TAM);
    }
    else {
        fscanf(arqTamanho, "%i", &TAM);
    }
    fclose(arqTamanho);
}
```

Gravando pessoas no arquivo

```
void gravaPessoas(Pessoa PESSOAS[]){

    FILE* arqPessoas = fopen(“pessoas.dat", "wb");
    fwrite(PESSOAS, sizeof(Pessoa), TAM, arqPessoas);
    fclose(arqPessoas);

    FILE* arqTamanho = fopen("tamanho.dat", "wb");
    fprintf(arqTamanho, "%i", TAM);
    fclose(arqTamanho);
}
```

Carregando pessoas do arquivo

```
void carregaPessoas(Pessoa PESSOAS[]) {

    FILE* arqPessoas = fopen(“pessoas.dat", “rb+");

    if(arqPessoas == NULL){
        arqPessoas = fopen(“pessoas.dat","wb+");
    }




}
```

Carregando pessoas do arquivo

```
void carregaPessoas(Pessoa PESSOAS[]) {

    FILE* arqPessoas = fopen(“pessoas.dat", “rb+");

    if(arqPessoas == NULL){
        arqPessoas = fopen(“pessoas.dat","wb+");
    }

     fread(PESSOAS, sizeof(Pessoa), TAM, arqPessoas);



}
```

Carregando pessoas do arquivo

```
void carregaPessoas(Pessoa PESSOAS[]) {

    FILE* arqPessoas = fopen(“pessoas.dat", “rb+");

    if(arqPessoas == NULL){
        arqPessoas = fopen(“pessoas.dat","wb+");
    }

     fread(PESSOAS, sizeof(Pessoa), TAM, arqPessoas);

     fclose(arqPessoas);

}
```

Uma possível função principal

```
int main() {
    setlocale(LC_ALL, "");
    abertura();
    Pessoa PESSOAS[MAX];









return 0;
}
```

Uma possível função principal

```
int main() {
    setlocale(LC_ALL, "");
    abertura();
    Pessoa PESSOAS[MAX];
    abreArquivoTamanho();
    carregaPessoas(PESSOAS);

return 0;
}
```

Uma possível função principal

```
int main() {
    setlocale(LC_ALL, "");
    abertura();
    Pessoa PESSOAS[MAX];
    abreArquivoTamanho();
    carregaPessoas(PESSOAS);
    int OPCAO;
    do{     OPCAO = menu();









    } while(OPCAO!=0);
    return 0;
}
```

Uma possível função principal

```
int main() {
    setlocale(LC_ALL, "");
    abertura();
    Pessoa PESSOAS[MAX];
    abreArquivoTamanho();
    carregaPessoas(PESSOAS);
    int OPCAO;
    do{     OPCAO = menu();
            switch(OPCAO){
              case 0 : despedida();
                       gravaPessoas(PESSOAS);
                       break;









            }
    } while(OPCAO!=0);
    return 0;
}
```

Uma possível função principal

```
int main() {
    setlocale(LC_ALL, "");
    abertura();
    Pessoa PESSOAS[MAX];
    abreArquivoTamanho();
    carregaPessoas(PESSOAS);
    int OPCAO;
    do{     OPCAO = menu();
            switch(OPCAO){
              case 0 : despedida();
                       gravaPessoas(PESSOAS);
                       break;
              case 1 : cadastrePessoa(PESSOAS);
                       break;



            }
    } while(OPCAO!=0);
    return 0;
}
```

Uma possível função principal

```
int main() {
    setlocale(LC_ALL, "");
    abertura();
    Pessoa PESSOAS[MAX];
    abreArquivoTamanho();
    carregaPessoas(PESSOAS);
    int OPCAO;
    do{     OPCAO = menu();
            switch(OPCAO){
              case 0 : despedida();
                       gravaPessoas(PESSOAS);
                       break;
              case 1 : cadastrePessoa(PESSOAS);
                       break;
              case 2 : listaPessoas(PESSOAS);
                       break;

            }
    } while(OPCAO!=0);
    return 0;
}
```

Uma possível função principal

```
int main() {
    setlocale(LC_ALL, "");
    abertura();
    Pessoa PESSOAS[MAX];
    abreArquivoTamanho();
    carregaPessoas(PESSOAS);
    int OPCAO;
    do{     OPCAO = menu();
            switch(OPCAO){
              case 0 : despedida();
                       gravaPessoas(PESSOAS);
                       break;
              case 1 : cadastrePessoa(PESSOAS);
                       break;
              case 2 : listaPessoas(PESSOAS);
                       break;
              default: printf("\n\aOpção inválida");
            }
    } while(OPCAO!=0);
    return 0;
}
```

Em discussão:

- Tipo primitivo
- Tipo Definido pelo Usuário
- Tipo Abstrato de Dados

Tente: tipo Data

Defina um tipo para representar datas: dd/mm/aaaa

Trabalho

Ao descrever uma pessoa, substitua o dado relativo à idade por sua data de nascimento.