Operadores e Funções String

Operadores

- char strcat(s1,s2)
- char *s1,*s2;
- copia a string s2 para o final de s1.

- char strncat(s1,s2,n)
- char *s1,*s2;
- copia no caractere de s2 para o final de s1.

- char strcmp(s1,s2)
- char *s1,*s2;
- compara duas strings retomando
- 0- se iguais
- 1- se a 1ª. for maior que 2ª.
- 2- se a 2ª. for maior que 1ª.

- char strcpy(s1,s2)
- char *s1,*s2;
- copia a string s2 para a string s1

- strlen(s)
- char *s;
- retorna o total de caracteres de s, exceto o caracter nulo
- atoi(s)
- char *s;
- converte string em número inteiro

Podemos "truncar" variáveis tipo string, procedendo como se estas fossem matrizes de caracter.

Exemplo: Escreva o nome do mês correspondente ao número.

```
main()
{
int i,j,k;
char *mes = "JANFEVMARABRMAIJUNJULAGOSETOUTNOVDEZ";
cls();
printf("Digite o Mes: ");
scanf("%d",&j);
k=j*3-3;
for(i=k;i<k+3;i++)
printf("%c",mes[i]);
}
```

No Exemplo a seguir, determinamos o exato tamanho da variável digitada, através da função "strlen", e "misturamos" os tipos "tot" e "a" de forma conseqüente.

Exemplo: Determine o "número da sorte"de um nome.

```
main()
{
int tot=0,i,w;
char a,s[20];
cls();
printf("Digite seu Nome: "); scanf("%s",s);
for(i=0;i<=strlen(s)-1;i++) {
        a = s[i];
        tot = tot + a - 64;
  }
printf("%s seu numero da sorte e' %d",s,tot);
}
```

Podemos também converter strings e números (inteiros/fracionários) conforme desejarmos:

- Exemplo: Conversão de String em Número Inteiro

```
main()
{
int i;
char s[10];
printf("Digite uma sequencia de numeros com letras: ");
gets(s);
i = atoi(s);
printf("Numero: %d ",i);
}
```

Laboratório (Aula 19L)

1- Elabore um programa que armazene e exiba notas de alunos de classes de no máximo 4 alunos. Existem apenas 3 classes de alunos.

2- Dado um nome, inverta-o.

3- Dado um nome, ordene suas letras de forma crescente.

Entradas e Saídas em Dispositivos

As funções e programas a seguir foram concebidos para serem compilados pelo Turbo C, portanto algumas adaptações poderão ser necessárias para sua utilização no Classic C. Entretanto a maior parte dos programas funcionará de forma adequada também no Classic C.

Nome	**Função**
fclose()	Fecha uma fila
feof()	Devolve se fim de fila
ferror()	Devolve Verdadeiro se um erro tiver ocorrido
fopen()	Abre uma fila
fprint()	Saída
fscanf()	Entrada
fseek()	Procura um byte especificado na fila
getc()	Lê um caracter na fila
putc()	Grava um caracter na fila
remove()	Apaga o arquivo
rewind()	Reposiciona o ponteiro do Arquivo em seu início

Para podermos utilizar estas instruções, temos que carregar a biblioteca "stdio.h", que contém estas rotinas.

Sintaxes:

- fopen()

A função "fopen" tem duas finalidades, a saber:

- abrir uma fila de bytes
- ligar um arquivo em disco àquela fila

```
FILE *fopen(char *NomeArquivo, char *modo);
```

Exemplo: Abertura de arquivo para gravação:

```
if ((fp = fopen("teste","w")) = NULL) {
        puts("Não posso abrir o Arquivo teste.\n");
        exit(1); /* força o término da execução da rotina */
  }
```

A tabela a seguir apresenta os modos de abertura de arquivo válidos:

Modo	**Significado**
"r"	Abre Arquivo de Texto para Leitura
"w"	Cria Arquivo de Texto para Gravação
"a"	Anexa a um Arquivo de Texto
"rb"	Abre Arquivo Binário para Leitura
"wb"	Cria Arquivo Binário para Gravação
"ab"	Anexa a um Arquivo Binário
"r+"	Abre Arquivo de Texto para Leitura/Gravação
"w+"	Cria Arquivo de Texto para Leitura/Gravação
"a+"	Abre ou Cria Arquivo de Texto para Leitura/Gravação
"r+b"	Abre Arquivo Binário para Leitura/Gravação
"w+b"	Cria Arquivo Binário para Leitura/Gravação
"a+b"	Abre ou Cria Arquivo Binário para Leitura/Gravação
"rt"	Idem a "r"
"wt"	Idem a "w"
"at"	Idem a "a"
"r+t"	Idem a "r+"
"w+t"	Idem a "w+"
"a+t"	Idem a "a+"

- putc()

Grava caracteres em fila previamente abertos

```
int putc(int ch, FILE *fp);
```

- ch é o caracter a ser gravado
- fp é o ponteiro devolvido por fopen

- getc()

Ler caracteres em uma fila aberta

```
int getc(FILE *fp);
```

Exemplo:

```
ch = getc(fp);
while (ch != EOF)
        ch = getc(fp);
```

- fclose()

Fechar as filas abertas. Caso o programa seja encerrado sem que as filas sejam fechadas, dados gravados nos buffers podem ser perdidos.

```
int fclose(FILE *fp);
```

- ferror()

Determina se a operação de arquivo produziu um erro. Sua forma geral será:

```
int ferror(FILE *fp);
```

- rewind()

Reinicia o arquivo, equivale ao Reset do Pascal, ou seja apenas movimenta o ponteiro do arquivo para seu início.

Exemplo (Funciona no Classic C) de montagem de um pequeno cadastro de nomes, endereços e salários de funcionários.

```
struct registro {
        char nome[40];
        char endereco[40];
        float valor;
} matriz[100];

main()
{
char escolha;
inicia_matriz();
for (;;) {
        escolha = menu();
        switch (escolha) {
                case 'i' :
                        inserir();
                        break;
                case 'e' :
                        exibir();
                        break;
                case 'c' :
                        carga();
                        break;
                case 's' :
                        salvar();
                        break;
                case 'f' :
                        saida();
           }
        }
puts(escolha);
}

saida()
{
cls();
exit();
}

inicia_matriz()
{
```

```
int t;
for (t=0;t<100;t++)
        *matriz[t].nome = '\0';
}

menu()
{
char s;
cls();
do {
        puts("Inserir");
        puts("Exibir");
        puts("Carregar");
        puts("Salvar");
        puts("Finaliza");
        printf("Digite a 1ª. Letra: ");
        scanf("%c",&s);
} while(s != 'i' && s != 'e' && s != 'c' && s != 's' && s != 'f');
return(s);
}

inserir()
{
int i;
for (i=0; i < 100; i++)
        if (!*matriz[i].nome) break;
                if (i==100) {
                                puts("Arquivo Cheio!");
                                return;
                }
                printf("Nome: "); gets(matriz[i].nome);
                printf("End.: "); gets(matriz[i].endereco);
                printf("Val.: "); scanf("%f",&matriz[i].valor);
}

exibir()
{
char x;
int t;
cls();
for(t=0;t<100;t++) {
        if (*matriz[t].nome) {
                printf("%s \n", matriz[t].nome);
                printf("%s \n", matriz[t].endereco);
                printf("%f \n", matriz[t].valor);
                puts(" "); puts("<Enter> para prosseguir!");
                x = getchar();
                }
        else
                break;
         }
}
salvar()
{
FILE *fp;
int i;
if ((fp=fopen("LISTA.DAT","wb"))==NULL) {
        puts("Falhou Abertura! ");
        return;
 }
for (i=0;i<100;i++)
        if (*matriz[i].nome)
                if (fwrite(&matriz[i],sizeof(struct registro), 1,fp) != 1)
                        puts("Falha na Gravacao! ");
        fclose(fp);
}

carga()
{
FILE *fp;
int i;
if ((fp=fopen("LISTA.DAT","rb")) == NULL) {
        puts("Falha na Abertura do Arquivo!");
        return;
}
```

```
inicia_matriz();
for (i=0; i < 100; i++)
        if (fread(&matriz[i], sizeof(struct registro), 1, fp) != 1) {
                if (feof(fp)) {
                        fclose(fp);
                        return;
                }
        else {
                puts("Erro de Leitura! ");
                fclose(fp);
                return;
     }
    }
}
```

Laboratório (Aula 20L)

- 1- Escreva um programa que armazene números primos numa matriz.
- 2- Escreva um nome, num arquivo.
- 3- Retome o exercício 1, porém armazene-o num arquivo.

Operações com Arquivo

Através dos exemplos que serão apresentados a seguir, constataremos a boa interação existente entre a linguagem C e o sistema operacional, antes porém apresentaremos uma nova forma de escrever "main()".

Vamos supor que desejamos criar um programa que escreva num arquivo cujo nome será fornecido na chamada do programa (Exemplificando: KTOD TESTE <Enter>). Gostaríamos que o DOS criasse o arquivo TESTE guardando o conteúdo digitado durante a execução do programa.

Para isso temos a seguinte sintaxe para "main()":

```
main(argv,argc)
```

onde

argc - tem o número de argumentos contidos nas linha de comando (necessariamente maior ou igual a um, pois o próprio programa já é considerado um argumento pelo D.O.S.). Argv é um ponteiro que acomodará os caracteres digitados.

Exemplo 1: Programa KTOD, que escreve caracteres num arquivo criado/aberto via D.O.S.

```
#include "stdio.h"
main(argc,argv)
int argc;
char *argv[];
{
FILE *fp;
char ch;
if (arg != 2) {
        printf("Digite o Nome do Arquivo\n");
        exit(1);
        }
if ((fp=fopen(argv[1],"w")) == NULL) {
        printf("Arquivo não pode ser aberto\n");
        exit(1);
         }
do {
        ch = getchar();
        putc(ch,fp);
        } while( ch != '$');
fclose(fp);
```

```
}
```

Exemplo 2: Programa DTOV, que apresenta em vídeo os caracteres digitados via KTOD.

```
#include "stdio.h"
main(argc,argv)
int argc;
char *argv[];
{
FILE *fp;
char ch;
if (arg != 2) {
        printf("Digite o Nome do Arquivo\n");
        exit(1);
         }
if ((fp=fopen(argv[1],"w")) == NULL) {
        printf("Arquivo não pode ser aberto\n");
        exit(1);
        }
ch = getc(fp);
while (ch != EOF) {
        putchar(ch);
        ch=getc(fp);
} while( ch != '$');
fclose(fp);
}
```

Laboratório (Aula 21L)

1- Uso do Compilador Turbo C. Testar exemplos vistos em teoria.

2- Elabore programa que exiba um programa em C armazenado em disco (a semelhança do comando type do DOS).

3- O que ocorre com o arquivo DADOS.DAT (que contém dados) após a execução do seguinte programa.

```
main()

{

FILE *fp;

if ((fp=fopen("DADOS.DAT","w"))==Null) {

exit(1);

}

fclose(fp);

}
```

Operações com Arquivo

Exemplo 1: Programa para copiar Arquivos.

```
#include "stdio.h"
main(argc,argv)
int argc;
```

```
char *argv[];
{
FILE *in, *out;
char ch;
if (arg != 3) {
        printf("Digite o Nome dos Arquivos\n");
        exit(1);
        }
if ((in=fopen(argv[1],"rb")) == NULL) {
        printf("Arquivo origem não existe\n");
        exit(1);
        }
if ((out=fopen(argv[2],"wb")) == NULL) {
        printf("Arquivo destino não existe\n");
        exit(1);
        }
while (! feof(in))
        putc(getc(in),out);  /* esta é a cópia propriamente dita */
fclose(in);
fclose(out);
}
```

Outras funções de tratamento de Arquivos.

- getw() e putw()

Idênticas a getc() e putc(), porém trabalhando com inteiros.

- fgets() e fputs()

Sintaxes:

```
char *fputs(char *str, FILE *fp);
char *fgets(char *str, int comprimento, FILE *fp);
```

fputs() é análoga a puts(), porém escreve em disco.

fgets() lê uma "string" da fila especificada, incluindo caracteres como

\n, porém a "string" lida sempre será finalizada com zero ASCII.

- fread() e fwrite()

Permitem que leiamos/gravemos blocos de dados, sua forma geral é a seguinte:

```
int fread(void *buffer, int num_bytes, int cont, FILE *fp);
int fwrite(void *buffer, int num_bytes, int cont, FILE *fp);
```

Exemplo 2: Leitura de Arquivos contendo números.

```
main()
{
FILE *fp;
float f = 12.23;
if ((fp=fopen("teste","wb")) == NULL) {
        printf("Arquivo não pode ser criado\n");
        exit(1);
         }
fwrite(&f,sizeof(float(),1,fp);
fclose(fp);
}
```

- fseek()

Entrada e saída com acesso aleatório

```
int fseek(FILE *fp, long int num_bytes, int origem);
```

fp - é o ponteiro de arquivo devolvido por fopen().

num_bytes - é um inteiro longo que representa o número de bytes desde a origem até chegar a posição corrente.

Este comando é normalmente utilizado em arquivos binários.

Exemplo 3: Leitura de um caracter em um arquivo binário.

```
main()
{
FILE *fp;
if ((fp=fopen("teste","rb")) == NULL) {
        printf("Arquivo não pode ser aberto\n");
        exit(1);
        }
fseek(fp,234L,0);      /* L força que seja um inteiro longo */
return getc(fp);       /* lê o caracter 234 */
}
```

Exemplo 4: Programa EDL

```
include <stdio.h>
main()
{
int c;
while ((c = getchar()) != EOF)
        putchar(c);
 }
```

Laboratório (Aula 22L)

- 1- Compile e Estude o programa EDL.
- 2- Construa os Programas KTOD e DTOV usando o Turbo C.
- 3- Construa o comando COPIAR, visto na teoria.
- 4- Elabore programa para criação de um dump de arquivo.

Operações com Arquivo

Caso o programador desejasse reproduzir entradas e saídas semelhantes as existentes em teclado e vídeo, ele poderia utilizar os comandos "fprintf" e "fscanf" conforme as sintaxes a seguir:

```
fprintf(fp,"string de controle",lista de argumentos);
fscanf(fp,"string de controle",lista de argumentos);
```

A linguagem C permite a criação de 5 tipos de estruturas particulares, a saber:

- Estrutura, um grupo de variáveis sobre o mesmo nome.
- Campo de Bit, que permite fácil acesso a bits dentro de uma palavra.
    - União, que permite que definamos a mesma parte da memória contendo dois ou mais tipos de variáveis (obviamente não usadas concomitantemente).
- Enumeração, que é similar a uma lista de símbolos.
- Tipo Final, onde o programador cria um novo nome para um tipo já existente.

A seguir criaremos um registro que fará uso destes conceitos. Supondo que desejamos um registro contendo Nome, Endereço, etc., teríamos:

```
struct regs {
        char nome[30];
        char rua[40];
        char cidade[20];
        char estado[02];
        unsigned long int cep;
}; /* ponto e vírgula encerrando o registro */
```

Dados

Além dos tipos básicos de dados que costumamos utilizar em nossos exemplos (int, float, char, etc.), em C dispomos de modificadores que podem mudar a forma de armazenar e de acessar determinada variável.

Modificadores de Acesso e de Armazenamento

O modificador const, impede que uma variável mude de valor durante a execução do programam conforme segue:

```
const float ver=3.20;
```

neste caso *ver* não poderá ser alterada durante a execução do programa.

O modificador volatile, permite que determinada variável tenha seu valor alterado sem um comando do programa. Este tipo de situação ocorre por exemplo no caso de seu programa acessar o relógio da máquina. Neste caso sua variável de tempo será atualizada constantemente sem que seja necessária qualquer instrução de seu programa.

O modificador auto, raramente usado, serve para declarar variáveis locais. O modificador extern permite que variáveis globais declaradas em módulos compilados separadamente possam ser adequadamente utilizadas.

Exemplo:

Prog. Um

```
int x,y;
main()
{
  .
  .
 }
func1()
{
        }
```

Prog. Dois

```
extern x,y;
funcDois()
{
   .
   .                          x = y++;
   .
 }

x = 12;         y = 3;
```

O modificador static permite que uma variável seja permanente dentro de sua própria função. Diferentemente da variável global, esta variável somente é válida em sua própria rotina.

Exemplo:

```
serie()
{
static int ser_num;
ser_num = ser_num + 12;
return(ser_num);
}
```

No exemplo anterior ser_num continua a existir entre as chamadas das funções ao invés de existir somente dentro da função *série*, como ocorreria com uma variável local normal.

O modificador register permite, caso seja possível, que determinada variável inteira ou caracter, não se localize na memória convencional, fique num registrador disponível (se houver um) da CPU, tornando a execução do programa muito mais veloz. Basta para isso declaramos:

```
register int i;
for(i=0;i<30000;i++)
```

Caso seja possível o contador i será processado muito mais rapidamente do que seria sem o modificador register.

Conversões de Tipos

O exemplo a seguir mostra a facilidade que temos para converter tipos em C.

```
main()
{
int x=70,x1=7;
char c='a',c1='A';
float f=23.215;
printf("x= %d, c= %c, f= %2.2f, x1= %d, c1= %c \n",x,c,f,x1,c1);
c = x;
printf("c <--x : c= %c\n",c);
x = f;
printf("x <--f : x= %d\n",x);
f = x1;
printf("f <--x1: f= %2.2f\n",f);
f = c1;
printf("f <--c1: f= %2.2f",f);
}
```

```
x= 70, c= a, f= 23.22, x1= 7, c1= A
c <--x : c= F
x <--f : x= 23
f <--x1: f= 7.00
f <--c1: f= 65.00
```

Ponteiros de funções

Da mesma forma como uma variável pode ser "apontada" por uma variável tipo ponteiro, uma função também poderá ser. Imaginemos um programa onde desejássemos comparar duas palavras numa função que as recebesse juntamente com um ponteiro que indicasse o sucesso ou não da comparação. No programa principal a função teria que ser declarada como um valor e depois poderia ser apontada por um ponteiro. Particularmente neste caso temos poucas vantagens em usar este raciocínio, porém em alguns casos poderemos usar este método de maneira a melhorarmos nossos programas.

Estruturas

Em C, uma estrutura é uma coleção de variáveis referenciadas através de um nome definido pelo programador, semelhante ao record da linguagem Pascal.

Exemplo de programa com um pequeno cadastro:

```
struct dados {
char nome[30];
char rua[40];
char cidade[20];
char estado[02];
char cep[09];
  };
```

Não devemos confundir tipo com variável, de forma que é errado afirmar-se coisas como "a variável dados recebeu ...", este tipo (dados) servirá para posteriormente declaramos uma variável como segue:

```
struct dados cliente;
ou ainda
struct dados {
char nome[30];
char rua[40];
char cidade[20];
char estado[02];
char cep[09];
} cliente;
```

Matrizes de Estrutura

Vamos supor termos definido um registro. Será que nos bastará podermos manipular um registro por vez, ou seria mais interessante termos como manipular uma série deles ao mesmo tempo? Para tornar isto possível devemos ter uma matriz que reflita a estrutura declarada.

Exemplo:

```
struct dados fornecedor[100];
```

Este trecho de código criaria uma matriz (vetor) com 100 elementos organizados pelo formato de dados.

Construção de Programa com acesso a registros

A seguir iremos contruir um exemplo completo de programa que acessa um tipo especial de arquivo, ou seja, aquele que contém apensas registros. Basicamente a teoria de Banco de Dados é fundamentada nos conceitos que estão sendo exemplificados a seguir.

```
struct registro {
int codigo;
long qtd;
double valor;
char descric[20];
};

main()
{
struct registro r, *rp;
r.codigo = 999;
r.qtd = 113333L;
r.valor = 23.21e6;
strcpy(r.descric,"MOTOR DE EXPLOSAO");
printf("Codigo: %d, Qtd: %ld, Valor: %.2f\n",r.codigo,r.qtd,r.valor);
printf("Descricao: %s\n",r.descric);
rp = &r;
printf("Codigo: %d, Qtd: %ld, Valor: %.2f\n",rp->codigo,rp->qtd,rp->valor);
strcpy(rp->descric,"MOTOR DE ARRANQUE");
printf("Descricao: %s\n",rp->descric);
}
```

Laboratório - Arquivos (Aula 23L)

- 1. No programa a seguir o carregaremos de um arquivo para uma matriz em memória um arquivo com dados de clientes, que nos permitirá inserir novos nomes, exibir os dados armazenados em memória e novamente armazená-lo em disco.

```
struct registro {
        char nome[40];
        char endereco[40];
        float valor;
```

```
} matriz[100];

main()
{
char escolha;
inicia_matriz();
for (;;) {
        escolha = menu();
        switch (escolha) {
                case 'i' :
                        inserir();
                        break;
                case 'e' :
                        exibir();
                        break;
                case 'c' :
                        carga();
                        break;
                case 's' :
                        salvar();
                        break;
                case 'f' :
                        saida();
                }
        }
        puts(escolha);
}

saida()
{
cls();
exit();
}
inicia_matriz()
{
int t;
for (t=0;t<100;t++)
        *matriz[t].nome = '\0';
}

menu()
{
char s;
cls();
do {
        puts("Inserir");
        puts("Exibir");
        puts("Carregar");
        puts("Salvar");
        puts("Finaliza");
        printf("Digite a 1ª. Letra: ");
        scanf("%c",&s);
        } while(s != 'i' && s != 'e' && s != 'c' && s != 's' && s != 'f');
return(s);
}

inserir()
{
int i;
for (i=0; i < 100; i++)
        if (!*matriz[i].nome) break;
                if (i==100) {
                        puts("Arquivo Cheio!");
                        return;
                 }
        printf("Nome: "); gets(matriz[i].nome);
        printf("End.: "); gets(matriz[i].endereco);
        printf("Val.: "); scanf("%f",&matriz[i].valor);
}

exibir()
{
char x;
int t;
cls();
```

```
for(t=0;t<100;t++) {
        if (*matriz[t].nome) {
                printf("%s \n", matriz[t].nome);
                printf("%s \n", matriz[t].endereco);
                printf("%f \n", matriz[t].valor);
                puts(" "); puts("<Enter> para prosseguir!");
                x = getchar();
              }
        else
                break;
         }
}

salvar()
{
FILE *fp;
int i;
if ((fp=fopen("LISTA.DAT","wb"))==NULL) {
        puts("Falhou Abertura! ");
        return;
        }
for (i=0;i<100;i++)
        if (*matriz[i].nome)
                if (fwrite(&matriz[i],sizeof(struct registro), 1,fp) != 1)
                        puts("Falha na Gravacao! ");
fclose(fp);
}

carga()
{
FILE *fp;
int i;
if ((fp=fopen("LISTA.DAT","rb")) == NULL) {
        puts("Falha na Abertura do Arquivo!");
        return;
        }
inicia_matriz();
for (i=0; i < 100; i++)
        if (fread(&matriz[i], sizeof(struct registro), 1, fp) != 1)
                if (feof(fp)) {
                        fclose(fp);
                        return;
                }
        else {
                puts("Erro de Leitura! ");
                fclose(fp);
                return;
             }
}
```

- 2. Monte Registro que exiba o mês escolhido pelo usuário.

Arquivos - Conceito de Chaves

O programa anterior demonstra como devemos proceder para Carregar um arquivo do disco para a Memória. Permite listarmos e inserirmos dados neste arquivo.

Finalmente permite salvarmos os dados para o Arquivo. Ficou faltando uma rotina de pesquisa, para podermos consultar, excluir e alterar dados, além de uma rotina permitindo listar os dados organizados, por exemplo alfabeticamente.

Montaremos uma rotina que permita a ordenação destes dados usando ponteiros para isto. Uma rotina de pesquisa binária também poderia ser elaborada bastando para isto a criação de rotina que permitisse a navegação dentro do modelo abaixo demonstrado:

```
  Botafogo                    Olaria
```

```
          Bangu        C.Grande   Fluminense      Vasco
```

Observe que qualquer pesquisa acharia o time pesquisado no máximo em 3 tentativas. Isto ocorre porque a estrutura acima está devidamente balanceada. Suponha porém que as entradas não ocorram na sucessão acima mostrada (Flamengo, Botafogo, Olaria, Bangu, C.Grande, Fluminense e Vasco), mas ocorram na seqüência apresentada a seguir: América, Corinthians, Guarani, Palmeiras, Portuguesa, Santos e São Paulo. Teríamos:

- América
- --- Corinthians
- --- Guarani
- --- Palmeiras
- --- Portuguesa
- --- Santos
- --- São Paulo

Desta forma não seria adequado desenvolvermos uma rotina de pesquise sem antes escrevermos uma rotina de "balanceamento" da estrutura acima, que possibilitam obtermos estruturas no conceito de árvore binária balanceada. Felizmente todos os gerenciadores de arquivo disponíveis, providenciam este tipo de tratamento, desobrigando o programador de aplicação da elaboração deste tipo rotina.

A seguir dois exemplos completos de estruturas manipuladas por ponteiros.

Exemplo 1: Cadastro acessado por ponteiros, com acesso seqüencial

(Registro 1,2,3,4,5,67,8,...,100; em 100 Registros).

```
struct addr
{
char nome[30];
char rua[40];
char cidade[18];
char estado[02];
char cep[09];
char lixao[10];
struct addr *next;    /* ponteiro da prox entrada */
struct addr *prior;   /* ponteiro p/ entrada anterior */
} lista;
struct addr *start;   /* primeiro da lista */
struct addr *last;    /* ultimo da lista */
struct addr *null;
main()
{
int choice;
null = (struct addr *)malloc(sizeof(lista));
do {
        choice = menu_select();
        switch(choice) {
                case 1 :
                        enter();
                        break;
                case 2 :
                        delete();
                        break;
                case 3 :
                        list();
                        break;
                case 4 :
                        achar();
                        break;
                case 5 :
                        save();
                        break;
                case 6 :
                        load();
                        break;
                case 7 :
                        exit(0);
```

```
         }
        } while(1);
}

menu_select()
{
char s[80];
int c;
printf("1. Inserir Nomes\n");
printf("2. Retirar Nomes\n");
printf("3. Listar  Nomes\n");
printf("4. Procurar Nome\n");
printf("5. Salvar Arquivo\n");
printf("6. Carregar Arquivo\n");
printf("7. Finalizar\n");
do {
        printf("\nSua opcao: ");
        gets(s);
        c = atoi(s);
        } while (c < 0 || c > 7);
return c;
}

enter()
{
struct addr *info;
char *malloc();
do {
        info = (struct addr *)malloc(sizeof(lista));
        if (info == 0){
                printf("\nMemoria Esgotada!");
                return;
          }
        inputs("Entre Nome: ",info->nome,30);
        if (!info->nome[0])
                break;
                inputs("Entre Rua: ",info->rua,40);
                inputs("Entre Cidade: ",info->cidade,18);
                inputs("Entre Estado: ",info->estado,02);
                inputs("Entre CEP: ",info->cep,10);
        if (start!=0) {
                last->next = info;
                info->prior = last;
                last = info;
                last->next = null;
                }
        else {
                start = info;
                start->next = null;
                last = start;
                start->prior = null;
                 }
        } while(1);
}

inputs(prompt,s,count)
char *prompt;
char *s;
int count;
{
char p[255];
do {
        printf(prompt);
        gets(p);
        if (strlen(p) > count)
                puts("Muito Grande!");
        } while (strlen(p) > count);
        strcpy(s,p);
}

delete()
{
struct addr *info, *find();
char s[255];
int volta;
```

```
inputs("Entre Nome: ",s,30);
info = find(s);
if (info) {
        if (start == info) {
                start = info->next;
                start->prior = null;
                }
        else {
                info->prior->next = info->next;
                if (info != last)
                        info->next->prior = info->prior;
                else
                        last = info->prior;
     }
        free(info);
  }
}

struct addr *find(nome)
char *nome;
{
struct addr *info;
info = start;
while (info && info != null) {
        if (!strcmp(nome,info->nome))
                return info;
        info = info->next;
         }
return null;
}

list()
{
int t;
char lixo;
struct addr *info;
info = start;
while (info && info != null) {
        display(info);
        scanf("%c",&lixo);
        info = info->next;
         }
printf("\n\n");
}

achar()
{
struct addr *info,*find();
char s[255];
inputs("Entre Nome: ",s,30);
info = find(s);
if (info != null)
        display(info);
else
        printf("Nao Encontrado!\n");
}

display(info)
struct addr *info;
{
printf("%s\n",info->nome);
printf("%s\n",info->rua);
printf("%s\n",info->cidade);
printf("%s\n",info->estado);
printf("%s\n",info->cep);
printf("\n\n");
}

save()
{
int t,size;
struct addr *info;
char *p;
FILE *fp;
if ((fp = fopen("LISTA.DAT","w")) == 0) {
```

```
        puts("Falhou a Abertura!");
        exit(0);
    }
printf("Salvando Arquivo\n");
size = sizeof(lista);
info = start;
while (info && info != null) {
        p = (char *)info; /* convesao p/ ponteiro de caracter */
        for (t=0;t<size-1;++t)
                putc(*p++,fp); /* salva um byte */
        info = info->next; /* proximo */
           }
putc(EOF,fp); /* marcando fim de arquivo ... opcional */
fclose(fp);
}
load()
{
int t,size;
struct addr *info,*temp;
char *p,*malloc();
FILE *fp;
if ((fp = fopen("LISTA.DAT","r")) == 0) {
        puts("Falhou Abertura");
        exit(0);
           }
printf("Carregando Arquivo\n");
size = sizeof(lista);
start = (struct addr *)malloc(size);
if (!start) {
        puts("Acabou Memoria! ");
        return;
         }
info = start;
p = (char *)info; /* ponteiro para caracter */
while ((*p++ = getc(fp)) != EOF) {
        for (t=0;t<size-2;++t)
                *p++ = getc(fp); /* carrega byte a byte */
                info->next = (struct addr *)malloc(size); /* aloca mais memoria */
                if (!info->next) {
                                printf("Memoria Esgotada!\n");
                                return;
                        }
        info->prior = temp;
        temp = info;
        info = info->next;
        p = (char *)info;
        if (info == null) break;
        }
free(temp->next);
temp->next = null;
last = temp;
start->prior = null;
fclose(fp);
}
```

Exemplo 2: Registro ordenados alfabeticamente através de ponteiros:

```
struct addr
{
char nome[30];
char rua[40];
char cidade[18];
char estado[02];
char cep[09];
char vago[10];
struct addr *next;    /* ponteiro da prox entrada */
struct addr *prior;   /* ponteiro p/ entrada anterior */
} lista;
struct addr *start;   /* primeiro da lista */
struct addr *last;    /* ultimo da lista */
struct addr *null;    /* endereco inicial */
struct addr *entra;   /* entrada atual */
char mostra = 'S';
```

```
main()
{
int choice;
cls;
printf("Exibir Situacao (Enter Exibe - n nao): "); scanf("%c",&mostra);
null = (struct addr *)malloc(sizeof(lista));
do {
        choice = menu_select();
        switch(choice) {
                case 1 :
                        enter();
                        break;
                case 2 :
                        delete();
                        break;
                case 3 :
                        list();
                        break;
                case 4 :
                        achar();
                        break;
                case 5 :
                        save();
                        break;
                case 6 :
                        load();
                        break;
                case 7 :
                        exit(0);
         }
        } while(1);
}

menu_select()
{
char s[80];
int c;
printf("1. Inserir Nomes\n");
printf("2. Retirar Nomes\n");
printf("3. Listar  Nomes\n");
printf("4. Procurar Nome\n");

printf("5. Salvar Arquivo\n");
printf("6. Carregar Arquivo\n");
printf("7. Finalizar\n");
do {
        printf("\nSua opcao: ");
        gets(s);
        c = atoi(s);
        } while (c < 0 || c > 7);
        return c;
}
enter()
{
struct addr *info;
struct addr *old;
int situacao;
char lixo;
char *malloc();
do {
        last = (struct addr *)malloc(sizeof(lista));
        if(last == 0) {
                printf("\nMemoria Esgotada!");
                return;
    }
        inputs("Entre Nome: ",last->nome,30);
        if (!last->nome[0])
                break;
        inputs("Entre Rua: ",last->rua,40);
        inputs("Entre Cidade: ",last->cidade,18);
        inputs("Entre Estado: ",last->estado,02);
        inputs("Entre CEP: ",last->cep,10);
        if(start!=0) {
                info = start;
                old = start;
```

```
                situacao = 0;
                while (situacao==0) {
                        if(mostra!='n') {
                                printf("Posicionado em %s  <Enter>\n",info->nome);
                                scanf("%c",&lixo);
       }
                        if(*info->nome>*last->nome) {
                                if(mostra != 'n') {
                                        printf("Posicionou (1) %s\n",info->nome);
                                        situacao = 1;
                                        break;
                     }
                                if(info==null) {
                                        if (mostra != 'n')
                                                printf("Fim de Arquivo (2) %s\n",last->nome);
                                        situacao = 2;
                                        break;
                     }
                        }
                        old = info;
                        info = info->next;
                        if(info->next==null) {
                                if(mostra != 'n') {
                                        printf("Aponta para Fim de Arquivo (3) %s\n",info->nome);
                                        situacao = 3;
                                        break;
                           }
                  }
          }
                if(situacao==1) {
                if(info==start) {
                last->next = info;
                info->prior = last;
                last->prior = null;
                start = last;
                }
else {
        last->next = info;
        info->prior = last;
        last->prior = old;
        old->next = last;
         }
 }
 if(situacao==2) {
        last->next = null;
        old->next = last;
        last->prior = old;
  }
  if(situacao==3) {
                last->next = null;
                info->next = last;
                last->prior = info;
         }
  }
 else {
                start = last;
                start->next = null;
                start->prior = null;
          }
 }while(1);
}

inputs(prompt,s,count)
char *prompt;
char *s;
int count;
{
char p[255];
do {
        printf(prompt);
        gets(p);
        if (strlen(p) > count)
                puts("Muito Grande!");
        } while (strlen(p) > count);
strcpy(s,p);
```

```
}

delete()
{
struct addr *info, *find();
char s[255];
int volta;
inputs("Entre Nome: ",s,30);
info = find(s);
if (info) {
        if (start == info) {
                start = info->next;
                start->prior = null;
                }
        else {
                info->prior->next = info->next;
                if (info != last)
                        info->next->prior = info->prior;
                else
                        last = info->prior;
                    }
        free(info);
        }
}

achar()
{
struct addr *info, *find();
char s[255];
int volta;
inputs("Entre Nome: ",s,30);
info = find(s);
if (info==null)
        printf("Nao Encontrado!\n");
else
        display(info);
}

struct addr *find(nome)
char *nome;
{
struct addr *info;
info = start;
while (info && info != null) {
        if (!strcmp(nome,info->nome))
                return info;
        info = info->next;
    }
return null;
}
list()
{
int t;
char lixo;
struct addr *info;
info = start;
while (info && info != null) {
        display(info);
        scanf("%c",&lixo);
        info = info->next;
        }
printf("\n\n");
}

display(info)
struct addr *info;
{
printf("\n\n***  E X I B I N D O   R E G I S T R O  ***\n");
printf("%s\n",info->nome);
printf("%s\n",info->rua);
printf("%s\n",info->cidade);
printf("%s\n",info->estado);
printf("%s\n",info->cep);
printf("\n\n");
}
```

```
save()
{
int t,size;
struct addr *info;
char *p;
FILE *fp;
if ((fp = fopen("ALFA.DAT","w")) == 0) {
        puts("Falhou a Abertura!");
        exit(0);
  }
printf("Salvando Arquivo\n");
size = sizeof(lista);
info = start;
while (info && info != null) {
        p = (char *)info; /* convesao p/ ponteiro de caracter */
        for (t=0;t<size-1;++t)
                putc(*p++,fp); /* salva um byte */
                if(info->next==null)
                        break;
                info = info->next; /* proximo */
  }
putc(EOF,fp); /* marcando fim de arquivo ... opcional */
fclose(fp);
}
load()
{
int t,size;
struct addr *info,*temp;
char *p,*malloc();
FILE *fp;
if ((fp = fopen("ALFA.DAT","r")) == 0) {
        puts("Falhou Abertura");
        exit(0);
  }
printf("Carregando Arquivo\n");
size = sizeof(lista);
start = (struct addr *)malloc(size);
if (!start) {
        puts("Acabou Memoria! ");
        return;
  }
info = start;
p = (char *)info; /* ponteiro para caracter */
while ((*p++ = getc(fp)) != EOF) {
        for (t=0;t<size-2;++t)
                *p++ = getc(fp); /* carrega byte a byte */
        info->next = (struct addr *)malloc(size); /* aloca mais memoria */
        if (!info->next) {
                printf("Memoria Esgotada!\n");
                return;
    }
        info->prior = temp;
        temp = info;
        info = info->next;
        if(info->next==null)
                break;
        p = (char *)info;
        if (info == null)
                break;
        }
        free(temp->next);
        temp->next = null;
        last = temp;
        start->prior = null;
        fclose(fp);
}
```

Teste de Cadastro (Aula 24L)

- 1. Digitação do Exemplo Acima.

1. Critique a forma de recuperação dos dados previamente digitados.

- 2. Montar programa que escreva Dados num arquivo.
- 3. Montar programa a semelhança do Comando Type.
- 4. Montar programa que escreva um texto num arquivo.
- 5. Montar programa que leia texto previamente armazenado num arquivo.