LINGUAGEM DE PROGRAMAÇÃO C

Prof. Marco Aurélio da Fonseca

Apostila #3

1. Gráficos

É possível em C utilizar o modo gráfico para criar uma interface mais amigável com o usuário. Existe uma série de funções que permitem a utilização do modo gráfico em C, como veremos a seguir. A utilização destas funções depende da inclusão do arquivo de cabeçalho graphics.h.

O programa abaixo ilustra como iniciar o modo gráfico em C.

Ex:

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
void main()
{
  int gdriver = DETECT, gmode, errorcode;
  /* inicia modo gráfico, não esquecer de ajustar o diretorio BGI */
  initgraph(&gdriver, &gmode, "c:\\tc\\bgi");
  if (graphresult() != grOk)
  {
    printf("Erro no modo grafico: %s\n", grapherrormsg(errorcode));
    getch();
    exit(1);
  }
  /* aqui inicia o desenho gráfico */
  line(0, 0, getmaxx(), getmaxy());
  /* encerra modo gráfico */
  getch();
  closegraph();
}
```

Algumas das principais funções gráficas são:

line(int x1, int y1, int x2, int y2);

Desenha uma linha originada em *x1,y1* e terminada em *x2,y2*

rectangle(int left, int top, int right, int bottom);

Desenha um retângulo não preeenchido, iniciando nas coordenadas *left,top* e terminando em *right,bottom*

bar(int left, int top, int right, int bottom);

Desenha um retângulo preeenchido, iniciando nas coordenadas *left,top* e terminando em *right,bottom.* A cor e estilo de preenchimento é determinada pela função setfillstyle

circle(int x, int y, int radius);

Desenha um círculo não preeenchido, tendo como ponto central as coordenadas *x,y* e tendo como dimensão o raio especificado em *radius*

setcolor(int color);

Define a cor a ser utilizada para desenho, conforme tabela abaixo:

Valor numérico	Cor
0	BLACK
1	BLUE
2	GREEN
3	CYAN
4	RED
5	MAGENTA
6	BROWN
7	LIGHTGRAY
8	DARKGRAY
9	LIGHTBLUE
10	LIGHTGREEN
11	LIGHTCYAN
12	LIGHTRED
13	LIGHTMAGENT
14	YELLOW
15	WHITE

setbkcolor(int color);

Define a cor de fundo a ser utilizada para desenho.

setfillstyle(int pattern, int color);

Define o padrão e cor de preenchimento para funções como bar. As constantes para *pattern* são:

Constante	Valor	Descrição
EMPTY_FILL	0	Preenche com a cor de fundo
SOLID_FILL	1	Preenchimento sólido
LINE_FILL	2	Preenche com ---
LTSLASH_FILL	3	Preenche com ///
SLASH_FILL	4	Preenche com ///, linhas grossas
BKSLASH_FILL	5	Preenche com \\\, linhas grossas
LTBKSLASH_FILL	6	Preenche com \\\
HATCH_FILL	7	Preenchimento Light hatch
XHATCH_FILL	8	Preenchimento Heavy crosshatch
INTERLEAVE_FILL	9	Preenchimento Interleaving line
WIDE_DOT_FILL	10	Preenchimento Widely spaced dot
CLOSE_DOT_FILL	11	Preenchimento Closely spaced dot
USER_FILL	12	Preenchimento definido pelo usuário

floodfill(int x, int y, int border);

Preenche uma área iniciada em *x,y* e delimitada pela cor especificada em *border*. Utiliza os parâmetros correntes de preenchimento especificados através de setfillstyle.

2. Listas Encadeadas

Esta classe de estruturas de dados foi inspirada numa forma de armazenamento bastante comum na nossa vida cotidiana, pois, desde eras bastante remotas, o homem utiliza-se da idéia de listar coisas com o objetivo de melhor organizá-las.

Baseado nesse conceito bastante simples (listagem) a estrutura de dados lista foi concebida, onde, cada elemento da lista armazena uma informação útil e uma, ou mais, referencia(s) para outro elemento (limitação do mundo computacional) e assim por diante, desta forma, com o auxílio dos ponteiros, pode-se acrescentar ou retirar elementos conforme a necessidade.

Listas encadeadas geralmente são utilizadas para criar vetores de extensão de

memória desconhecida, pela sua característica dinâmica, permitindo que o vetor (lista) tenha sempre o tamanho necessário.

2.1. Listas Simplesmente Encadeadas

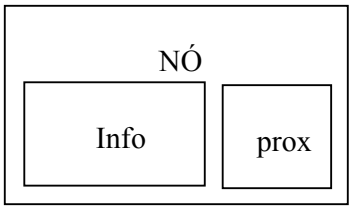
Este tipo de lista é o mais simples e segue as seguintes regras:

a. deve existir um ponteiro que aponte para o primeiro elemento da lista (início da lista);

b. cada elemento, ou nó, da lista aponta para o próximo sucessivamente (daí o nome listas encadeadas)

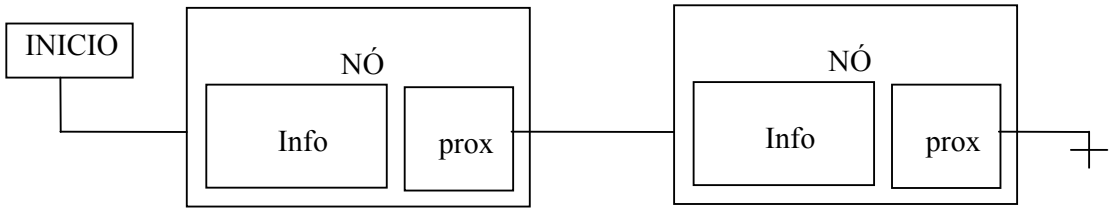
c. o último elemento deve apontar para NULL, indicando o final da lista.

Representação gráfica :



Onde: nó é um registro com dois campos: info e prox, info é um tipo registro do tipo infotype contendo as informações (variáveis) úteis e prox é um ponteiro capaz de armazenar o endereço de memória de um registro do tipo nó (elemento da lista).

Graficamente a representação de uma lista simplesmente encadeada seria a seguinte:



Onde: INICIO é um ponteiro que guarda o endereço de memória do primeiro elemento da lista.

2.1. Listas Duplamente Encadeadas

É uma lista encadeada, onde cada elemento contém um ponteiro para o nó anterior e um ponteiro para o nó posterior, permitindo o caminhamento nos dois

sentidos da lista, diferentemente da lista simplesmente encadeada, que só permitia em um único sentido.

3. Filas

Uma fila é uma lista linear de informação que é acessada na seguinte ordem, o "primeiro que entra, primeiro que sai" (FIFO). Uma fila não permite acesso a um elemento que não seja o primeiro.

Normalmente as filas são utilizadas na administração de recursos compartilhados, impondo uma prioridade por ordem de chegada. Um bom exemplo são as filas de impressão, onde, cada documento espera sua vez para ser impresso.

4. Pilhas

Análogo à estrutura fila, a estrutura pilha também pré-define a posição de inserção e remoção dos elementos, sendo que, a estratégia é "último que entra, primeiro que sai" (LIFO)

Um bom exemplo da aplicação das pilhas é a gerencia das chamadas as sub-rotinas utilizadas pela maioria dos compiladores das linguagens de programação.

Por razões históricas, as duas operações primárias de pilha - inserção e retirada - são normalmente chamadas de push e pop, respectivamente. Assim para implementar uma pilha são necessárias duas funções:

PUSH - coloca um elemento no topo da pilha

POP - retira um elemento do topo da pila

5. Árvores

Árvores são estruturas de dados essencialmente dinâmicas, permitindo a inserção, pesquisa e destruição de elementos.

Árvores trazem, implicitamente, a idéia de hierarquia, por isso são estruturas de dados muito utilizadas em gerenciamento de arquivos e validação de expressões.

Uma árvore pode ser definida de várias formas. Talvez, a maneira mais natural é defini-las recursivamente. Assim, uma árvore é composta por um conjunto de nós. Existe um nó denominado raiz, que contém zero ou mais sub-árvores e cada sub-árvore pode ser considerada uma árvore.

Nomenclatura:

Raiz - é o nó de mais alto nível da árvore (primeiro nó da árvore).

Nível - o número de nós que vai da raiz até um determinado nó.

Grau - o número de sub-árvores de um nó

Folha - nó com grau igual a zero

Altura - nível mais alto da árvore

Pai - a raiz de uma sub-árvore é pai das raízes de suas sub-árvores

Filho - os nós raízes de uma sub-arvores é filho da raiz da árvore

Irmão - nós raízes das sub-árvores são irmãos

5.1. Árvores Binárias

São árvores onde cada nó tem no máximo dois filhos (grau máximo 2). Com isto, obtém-se uma estrutura apropriada para busca binária. Outra área onde estas estruturas são muito utilizadas é na validação de expressões. Isto porque um operador relaciona apenas um ou dois operandos.

Uma propriedade importante de árvores binárias é que possuem profundidade (altura) bastante inferior ao número de nós.

A aplicação mais importante das árvores binárias é seu uso em busca binária. A cada nó da árvore associa-se uma chave que permite a realização de uma ordenação. A Construção da árvore deve ser de tal forma que na sub-árvore à esquerda de uma dada raiz só existam nós com chaves menores que a chave da raiz. E a sub-árvore à direita só pode conter nós com chaves maiores que a da raiz.