A MultiPilha – Pilha Estática Múltipla

Nesta estrutura, várias pilhas estáticas são implementadas sobre um único vetor particionado. Individualmente, cada partição contém uma pilha e cada pilha é associada ao respectivo descritor.

1. Implementação da Multipilha Estática – MpE

Cada célula do vetor que dá suporte à MpE compartilha diferentes tipos de dados em uma mesma região de memória, as células servem tanto à representação dos nós que referenciam dados quanto à representação de descritores. Em linguagem *C* é possível realizar esse tipo de compartilhamento através da estrutura do tipo união (*union*). Encare a MpE como uma boa oportunidade para exercitar esse recurso em *C*.

2. Utilizando a União - *Union*

Na implementação aqui proposta, são necessários os parâmetros N (correspondente ao número de pilhas) e L (relativo ao comprimento máximo de cada pilha) para a criação do vetor com N+N*L células do tipo união (espaço para os N descritores e suas respectivas pilhas de tamanho L). A fronteira "virtual" entre as regiões de descritores e a região das respectivas pilhas, é delimitada da seguinte forma: as N primeiras células do vetor servirão aos descritores enquanto as demais servirão aos nós que referenciam os dados das pilhas, conforme a Figura 1. Os valores N e L constarão em um descritor geral da MpE.

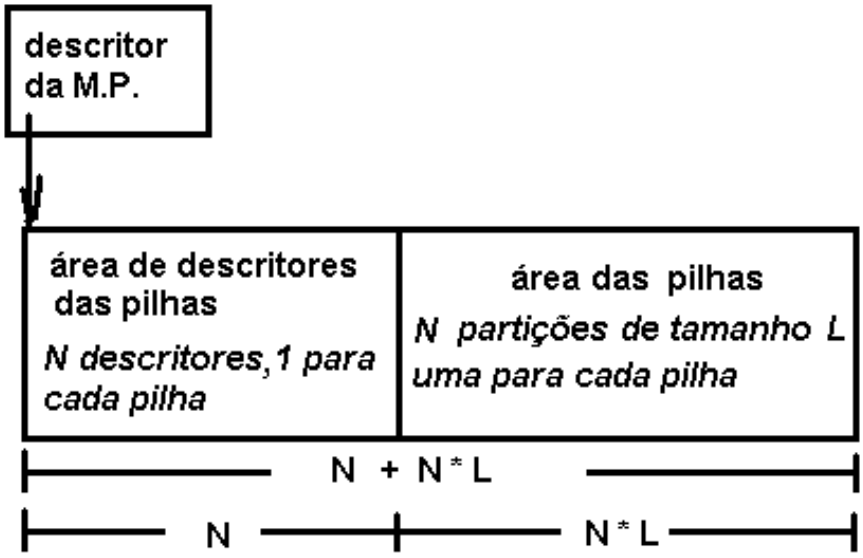

Figura 1 - Multipilha Estática.

3. Indexando uma Pilha na MpE

No modelo aqui discutido, o módulo de aplicação entende uma *multipilha* com N pilhas, como uma sequência começando da 1ª pilha, seguida da 2ª, até a *N-ésima* pilha. Não existe uma *pilha zero*.

Considerando-se a organização do vetor conforme descrito anteriormente e exibida na Figura 2, para acessar o *i*-ésimo descritor (1 ≤ i ≤ N) basta acessar ou indexar a célula *i-1* do vetor. Esse descritor conterá dados sobre a pilha (*i-ésima)* a ele associada.

Além de identificar o seu descritor, para realizar uma operação sobre uma pilha é preciso localizar as células do vetor correspondentes à mesma, ou seja, quais as células que delimitam o início da partição (*inicioPartição*) e o final da sua partição (*finalPartição*) da pilha-alvo em questão. A determinação desses valores é realizada com base nas informações no descritor dessa pilha, *inicioPartição* consta explicitamente no descritor. Quanto à segunda informação – *finalPartição* – poderá ser calculada facilmente pela expressão: *inicioPartição* + (*L* – 1).

Caso as informações de partição não existissem no descritor, ainda assim seria possível calcular o *inicioPartição* através da expressão (N+ L(i-1)), onde i corresponde à pilha-alvo e 1 ≤ i ≤ N.

A partir do valor *inicioPartição* se calcula *finalPartição* como sendo *inicioPartição*+L-1.

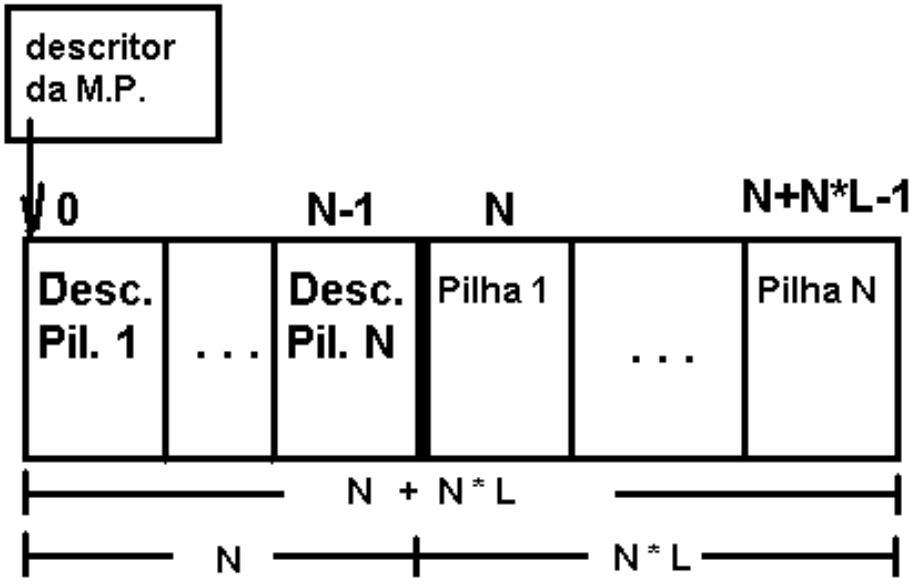

Figura 2 - Indexação dos descritores e respectivas pilhas.

4. Implementando a MpE.

Para a MpE a arquitetura de arquivos ainda é semelhante àquela utilizada para a implementação da PE e da PD, aqui, no entanto, eles serão nomeados como MpE.H, e MpE.C. Aqui não será exibido todo o código dos arquivos, apenas alguns fragmentos importantes para uma posterior implementação.

Diferentemente da *struct*, que ao ser instanciada reserva memória para todos os seus campos, ao instanciar uma *union* (ou união), reserva-se um espaço de memória que corresponde ao tamanho do maior campo definido, sendo que este espaço será compartilhado por todos os campos envolvidos na definição da *union*. Quando um desses campos é muito menor que os demais membros e é requisitado de maneira mais freqüente, tem-se um desperdício de memória. Dessa forma, em termos de um bom uso da memória, o uso de *union* só se justifica se houver um equilíbrio entre o tamanho (em *bytes*) dos elementos que constituirão a unidade da *union*. Isto está contemplado na implementação aqui proposta, onde a *union* definida como *NoMP* é constituída basicamente da struct info e da struct DescPilha.

Na Figura 3 é vista uma representação gráfica do resultado da criação da MpE. Observe que, por estarem vazias, as pilhas exibidas na figura possuem valores de *topo* que retratam tal estado. Para a Pilha Estática simples a representação do estado "pilha vazia" foi feita pela inicialização do *topo* com um valor que não indexa o vetor (utilizou-se a macro VAZIA que foi definida com o valor menos um "-1"). Para a MpE pretende-se utilizar a mesma concepção para iniciar os topos das N pilhas envolvidas. Outra possibilidade seria cada *topo* ser iniciado com um valor inteiro que não indexe a respectiva partição da pilha esse valor corresponderia ao início da respectiva partição menos um.

Fragmento do arquivo MPE.H

```
/* Nó descritor de uma pilha */
typedef struct {
    int topo;
    int inicioParticao;
} DescPilha;

/* Nó da Multi-Pilha UNION*/
typedef union {
    DescPilha descritor;
    info dados;
} NoMP;

/* Descritor da Multi-Pilha */
typedef struct mp {
    int N; /* N = Número de Pilhas*/
    int L; /* L = Tamanho máximo da partição de cada Pilha*/
    int tamInfo;
    NoMP *vet;
} MP;
```

Fragmento do arquivo MpE.C

```
MP* cria( MP *p, int N, int L, int tamInfo)
{
   int i, M;
   NoMP  *aux;
   MP *desc=NULL;

  if (N > 0 && L > 0 && tamInfo > 0)
   {    M = N*L;
        if((desc = (MP *) malloc(sizeof(MP))) !=NULL) /* 1 */
        {
              if( ( desc->vet = (NoMP *) malloc((M+N)* sizeof(NoMP)) ) == NULL) /* 2 */
             {    free (desc);
                   desc = NULL;
              }
              else
              {
                 desc->N = N;     /* 3 */
                 desc->L = L;
                 desc->tamInfo = tamInfo;
                 aux= desc->vet;      /* 4 */
                 for(i = 0; i < N ; i++)      /* 5 */
                {     (aux+ i)->descritor.topo = -1;     /* 6 */
                       (aux+ i)->descritor.inicioPartição = N +  i*L;
                }
              }
       }
       return desc;   /* 7 */
}

/*Comentários:

1. Criação do descritor geral da MpE;

2. Alocação de memória para a multipilha (vetor de unions) constituida  de M+N nós de
dados e os N nós descritores das respectivas pilhas;

3. Iniciando os atributos gerais da multi-pilha;

4. Ponteiro auxiliar aux apontando para a área dos descritores individuais;

5. Formatando a MpE: iniciando os descritores das pilhas;

6. Cada topo é iniciado com valor -1, indicando que a respectiva pilha está vazia;
Perceba que também é possível utilizar a sintaxe aux[i].descritor para acesso ao i-ésimo
descritor;

7. Sucesso na criação: após alocações e inicializações bem-sucedidas, o endereço do
descritor é retornado ao cliente. */
```

5. Acessando a MpE

Os exemplos descritos na Tabela 1 baseiam-se na formatação exibida na Figura 3, a qual representa uma MPE recém-criada segundo a função de criação anteriormente descrita. Considera-se a i-ésima pilha P_i onde $1 \leq i \leq N$ e que o apontador *ptr* contenha o endereço do descritor geral da MpE.

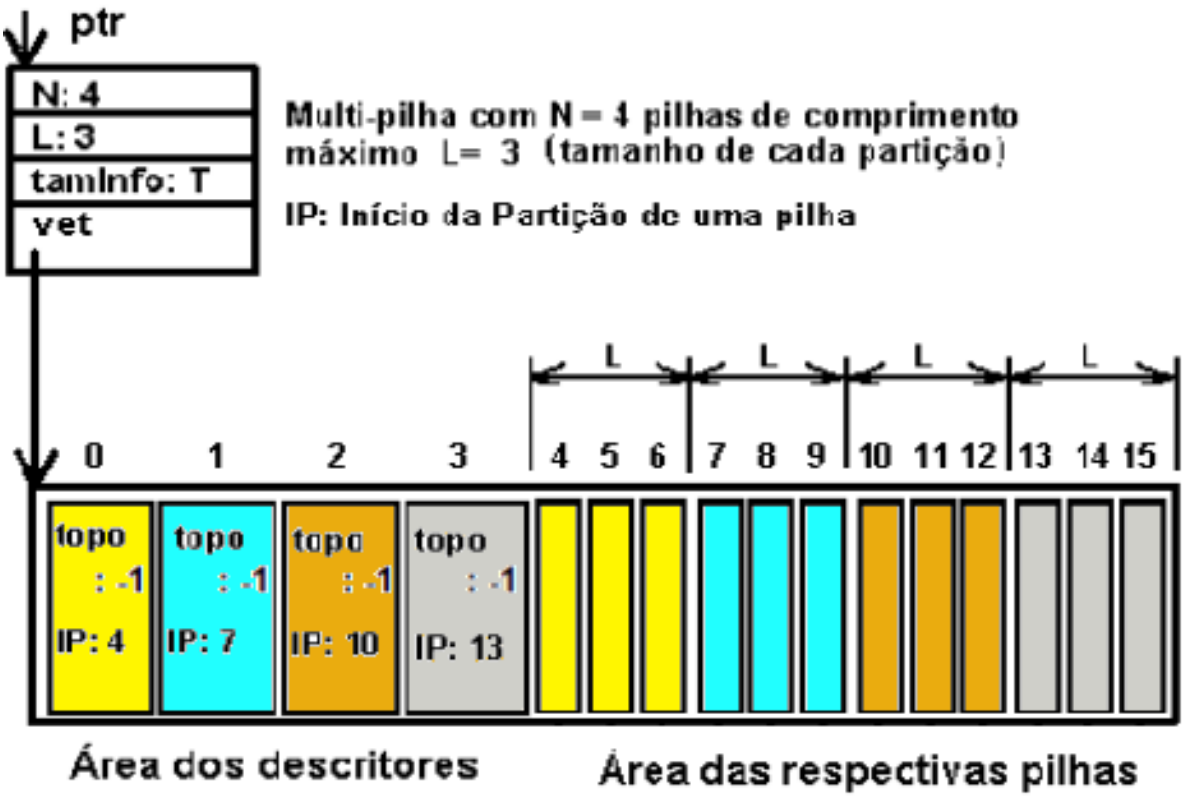

Figura 3- MPE recém criada para 4 pilhas de tamanho máximo 3.

Descrição	Fragmento
Pi corresponde à i-ésima pilha, onde 1 ≤ i ≤ N: 1ª pilha = P1, 2ª pilha = P2, ... última pilha = Pn Assim, o acesso ao i-ésimo descritor pode ser realizado por indexação utilizando i-1 como índice. Alternativa, por apontadores diretamente.	ptr->vet[i-1].descritor;
Acesso ao início da partição da i-ésima pilha.	IP_i = ptr->vet[i-1].descritor.inicioParticao;
Acesso ao final da partição da i-ésima pilha.	FP_i = IP_i + (ptr->L) -1;
Acesso ao topo da i-ésima pilha.	topo_i= ptr->vet[i-1].descritor.topo;
Índice referente ao topo da i-ésima pilha	IP_i+topo_i
Para buscar a informação no topo da i-ésima pilha (não vazia) basta indexar o vetor pelo valor do seu início de partição somado ao valor do seu topo. O topo é um deslocamento a partir do início da partição (-1 ≤ topo ≤ L-1).	enderecoOrigem = &(ptr->vet[IP_i+topo_i]); memcpy(enderecoDestino, enderecoOrigem , p->tamInfo);

Para empilhar na i-ésima pilha (não cheia) basta incrementar o seu topo e copiar os novos dados para a célula do vetor indexada pelo valor do seu início de partição somado ao valor do seu topo. O topo é um deslocamento a partir do início da partição (-1 ≤ topo ≤ L-1).	ptr->vet[i-1].descritor.topo += 1;
	topo_i=ptr->vet[i-1].descritor.topo;
	enderecoDestino = &(ptr->vet[IP_i+topo_i]);
	memcpy(enderecoDestino, novo, p->tamInfo);

Pata determinar o número de elementos empilhados na *i-ésima* pilha basta somar 1 ao valor atual do seu topo (-1 ≤ topo ≤ L-1).	numElementosPilha_i=ptr->vet[i-1].descritor.topo+1;
Sabendo que *-1 ≤ topo_i ≤ L-1*, para testar se a *i-ésima* pilha está vazia é necessário saber se seu topo é igual ao valor de inicialização: -1	SE (*topo_i == -1*) a pilha_i está vazia. SENÃO a pilha_i não está vazia
Sabendo que *-1 ≤ topo_i ≤ L-1*, o maior valor de topo da *i-ésima* pilha corresponde a *L-1* e ocorre quando a *i-ésima* pilha está cheia. Outras formas de teste dessa condição (cheia) consistem em saber, a partir do início da partição somado ao topo, se a pilha alcançou o final de sua partição. Outra forma, ainda, é verificar se o número de elementos empilhados é igual ao comprimento máximo da partição.	SE (*topo_i == (ptr->L) -1*) a pilha_i está cheia. SENÃO a pilha_i não está cheia Alternativas: SE (*IP_i+topo_i == FP_i*) a pilha_i está cheia. SENÃO a pilha_i não está cheia SE (*numElementosPilha_i == ptr->L*) a pilha_i está cheia. SENÃO a pilha_i não está cheia

Para apenas desempilhar a i-ésima pilha (não vazia) basta decrementar o seu topo.	ptr->vet[i-1].descritor.topo -=1;
Para reinicializar a i-*ésima* pilha basta atribuir menos um “-1” ao seu topo.	ptr->vet[i-1].descritor.topo = -1;

Tabela 1: Fragmentos de código para a MpE.

Bibliografia

[1] Horowitz, E. & Sahni, S. “Fundamentos de Estruturas de Dados”. Ed. Campus. 1984.