

Aula 3: Pilhas

Professor(a): Virgínia Fernandes Mota

<http://www.dcc.ufmg.br/~virginiaferm>

ALGORITMOS E ESTRUTURAS DE DADOS - SETOR DE INFORMÁTICA



- Uma das estruturas de dados mais simples é a pilha.
- Sua idéia fundamental é que todo o acesso a seus elementos seja feito a partir do topo.
- LIFO - Last in First Out
- Operações básicas: *push* (empilhar) e *pop* (desempilhar).

push(a)



push(b)



push(c)



pop(c)
retorna-se c



push(d)



- Consideraremos duas implementações de pilha: usando **vetor** e usando **lista encadeada**.
- Vamos considerar as seguintes operações:
 - criar uma pilha vazia;
 - inserir um elemento no topo (*push*);
 - remover o elemento do topo (*pop*);
 - verificar se a pilha está vazia;
 - liberar a estrutura de pilha;

Interface do tipo Pilha

- O arquivo *pilha.h* pode conter o seguinte código:

```
1 typedef struct pilha Pilha;  
2 Pilha *pilha_cria();  
3 void pilha_push(Pilha *p, float v);  
4 float pilha_pop(Pilha *p);  
5 int pilha_vazia(Pilha *p);  
6 void pilha_libera(Pilha *p);
```

- O tipo pilha criado pode ser implementado usando **vetor** ou **lista encadeada**

Implementação de pilha com vetor

- Em algumas aplicações computacionais é comum saber de antemão o número máximo de elementos que podem estar armazenados simultaneamente na pilha.
- Neste caso, a implementação da pilha pode ser feita por um vetor.

```
1 #define N 50;  
2 struct pilha{  
3     int n;  
4     float vet[N];  
5 };;
```

Implementação de pilha com vetor - Criação

```
1 Pilha *pilha_cria(){  
2     Pilha *p = (Pilha*) malloc (sizeof(Pilha));  
3     p->n = 0; //inicializa com zero elementos  
4     return p;  
5 }
```

Implementação de pilha com vetor - Inserção

```
1 void pilha_push(Pilha *p, float v){
2     if (p->n == N) {
3         printf("Capacidade da pilha estourou.");
4         exit(1);
5     }
6     //insere elemento na próxima posição livre
7     p->vet[p->n] = v;
8     p->n++;
9 }
```


Implementação de pilha com vetor - Remoção

```
1 float pilha_pop(Pilha *p){  
2     float v;  
3     if (pilha_vazia(p)){  
4         printf("Pilha vazia!");  
5         exit(1);  
6     }  
7     //Retira elemento do topo  
8     v = p->vet[p->n - 1];  
9     p->n--;  
10    return v;  
11 }
```

Implementação de pilha com vetor - Pilha vazia

```
1 int pilha_vazia(Pilha *p){  
2     return (p->n == 0);  
3 }
```

Implementação de pilha com vetor - Liberar

```
1 void pilha_libera(Pilha *p){  
2     free(p);  
3 }
```

Implementação de pilha com lista

- Quando o número máximo de elementos que serão armazenados na pilha não é conhecido, devemos implementar a pilha com uma estrutura de dados dinâmica.
- Usaremos então uma lista encadeada.

```
1 struct lista{  
2     float info;  
3     struct lista *prox;  
4 };  
5  
6 typedef struct lista Lista;  
7  
8 struct pilha{  
9     Lista *prim;  
10 };  
11
```

Implementação de pilha com lista - Criação

```
1 Pilha *pilha_cria(){  
2     Pilha *p = (Pilha*) malloc (sizeof(Pilha));  
3     p->prim = NULL;  
4     return p;  
5 }
```

Implementação de pilha com lista - Inserção

```
1 void pilha_push(Pilha *p, float v){  
2     Lista *n = (Lista*) malloc (sizeof(Lista));  
3     n->info = v;  
4     n->prox = p->prim;  
5     p->prim = n;  
6 }
```

Implementação de pilha com lista - Remoção

```
1 float pilha_pop(Pilha *p){  
2     Lista *t;  
3     float v;  
4     if (pilha_vazia(p)){  
5         printf("Pilha vazia");  
6         exit(1);  
7     }  
8     t = p->prim;  
9     v = t->info;  
10    p->prim = t->prox;  
11    free(t);  
12    return v;  
13 }
```

Implementação de pilha com lista - Pilha vazia

```
1 int pilha_vazia(Pilha *p){  
2     return (p->prim == NULL);  
3 }
```


Implementação de pilha com lista - Liberar

```
1 void pilha_libera(Pilha *p){  
2     Lista *q = p->prim;  
3     while (q != NULL){  
4         Lista *t = q->prox;  
5         free(q);  
6         q = t;  
7     }  
8     free(p);  
9 }
```

Implementação de pilha com lista - Impressão

```
1 //imprime versao com vetor
2 void pilha_imprime(Pilha *p){
3     int i;
4     for (i = p->n-1; i >= 0; i--)
5         printf("%f ", p->vet[i]);
6 }
7
8 //imprime versao com lista
9 void pilha_imprime(Pilha *p){
10     Lista *q;
11     for (q = p->prim; q != NULL; q = q->prox)
12         printf("%f ", q->info);
13 }
```

Estudo de caso: Calculadora pós-fixada

- Um bom exemplo de aplicação de pilha é o funcionamento das calculadoras HP.
- Elas trabalham com expressões pós-fixadas, então para avaliar uma expressão como $(1-2)*(4+5)$ podemos digitar $1\ 2\ -\ 4\ 5\ +\ *$.

Estudo de caso: Calculadora pós-fixada

```
1 struct calc{
2     char f[21];
3     Pilha *p; //pilha de operandos
4 }
5
6 typedef struct calc Calc;
7
8 Calc *calc_cria(char *formato){
9     Calc *c = (Calc*) malloc (sizeof(Calc));
10    strcpy(c->f, formato);
11    c->p = pilha_cria(); //cria pilha vazia
12    return c;
13 }
14
15 void calc_operando(Calc *c, float v){
16     pilha_push(c->p, v); //empilha operando
17     printf(c->f, v); //imprime topo
18 }
19
20 void calc_libera(Calc *c){
21     pilha_libera(c->p);
22     free(c);
23 }
```

Estudo de caso: Calculadora pós-fixada

```
1 void calc_operador( Calc *c, char op){
2     float v1, v2, v;
3     //desempilha operandos
4     if ( pilha_vazia(c->p))
5         v2 = 0.0;
6     else
7         v2 = pilha_pop(c->p);
8
9     if ( pilha_vazia(c->p))
10        v1 = 0.0;
11    else
12        v1 = pilha_pop(c->p);
13
14    //faz operacao
15    switch (op){
16        case '+': v = v1 + v2; break;
17        case '-': v = v1 - v2; break;
18        case '*': v = v1 * v2; break;
19        case '/': v = v1 / v2; break;
20    }
21
22    //empilha resultado
23    pilha_push(c->p, v);
24
25    //imprime topo da pilha
26    printf(c->f, v);
27 }
```

Estudo de caso: Calculadora pós-fixada

```
1 //Programa para ler expressão e chamar funções da calculadora
2 int main(){
3     char c;
4     float v;
5     Calc *calc;
6     calc = calc_cria("%.2f");// cria calculadora com formato de duas
7                                     casas decimais
8     do{
9         scanf(" %c", &c);
10        if (c == '+' || c == '-' || c == '*' || c == '/'){
11            calc_operador(calc, c);
12        }
13        else{
14            ungetc(c, stdin);
15            if (scanf("%f", &v) == 1)
16                calc_operando(calc, v);
17        }
18    } while (c != 'q');
19    calc_libera(calc);
20    return 0;
21 }
```

Estudo de caso: Calculadora pós-fixada

Exemplo de saída:

3 5 8 * + *digitado pelo usuário*

3.00

5.00

8.00

40.00

43.00

7 / *digitado pelo usuário*

7.00

6.14

q *digitado pelo usuário*

1. Implementar as funções/procedimentos apresentados em sala para a manipulação de pilhas. Crie *pilhas.c* e *pilhas.h* para a manipulação das pilhas (com vetor e com lista) e *main.c* para testes. Crie um *makefile* para compilar o código.
2. Faça um programa para determinar se a sequência de parênteses em uma expressão matemática está bem formada (ou seja, parênteses são fechados na ordem inversa àquela em que foram abertos).

Na próxima aula...

Exercícios para a prova.