3. Pilha

"A tecnologia é dominada por aqueles que
gerenciam o que não entendem."
Arthur Bloch

Uma pilha é um conjunto ordenado de itens, no qual novos itens podem ser inseridos e a partir do qual podem ser eliminados itens de uma extremidade, chamada topo da pilha. Também é chamada de lista linear, onde todas as inserções e eliminações são feitas em apenas uma das extremidades, chamada topo. A figura 3.1 mostra a representação de uma pilha.

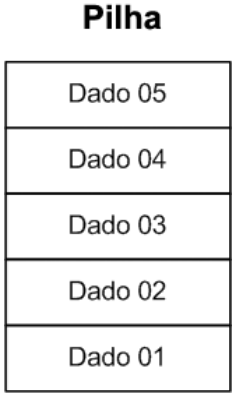

Figura 3.1: Exemplo de Pilha

A estrutura de dados do tipo pilha tem como característica que a última informação a entrar é a primeira a sair (LIFO - *last in first out*). A estrutura em pilha tem os seguintes métodos ou funções:

- *push* - coloca uma informação na pilha (empilha).
- *pop* - retira uma informação da pilha (desempilha).
- *size* - retorna o tamanho da pilha.
- *stackpop* - retorna o elemento superior da pilha sem removê-lo (equivalente às operações de pop e um push).
- *empty* - verifica se a pilha está vazia.

A aplicação da estrutura de pilhas é mais freqüente em compiladores e sistemas operacionais, que a utilizam para controle de dados, alocação de variáveis na memória etc.

O problema no uso de pilhas é controlar o final da pilha. Isto pode ser feito de várias formas, sendo a mais indicada criar um método para verificar se existem mais dados na pilha para serem retirados.

Tomando a pilha da figura 3.2 como exemplo, a operação **push(H)** irá acrescentar um novo elemento ao topo da pilha sendo, em seguida, executado um conjunto de operações sobre a pilha:

- 2 - push(I) - Coloca o elemento I no topo da Pilha
- 3 - pop() - Retorna o elemento I
- 4 - pop() - Retorna o elemento H
- 5 - pop() - Retorna o elemento F
- 6 - pop() - Retorna o elemento E
- 7 - pop() - Retorna o elemento D
- 8 - push(D) - Coloca o elemento D no topo da Pilha
- 9 - push(E) - Coloca o elemento E no topo da Pilha

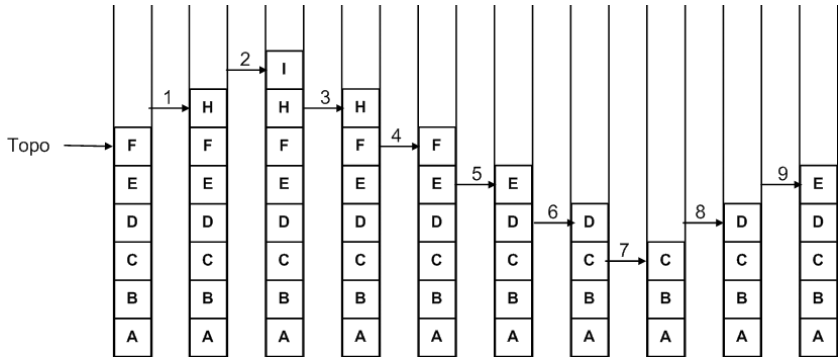

Figura 3.2: Operações em uma pilha

3.1 Representação das Operações com Pseudo-código

As operações de pilha podem ser representadas com algumas linhas de pseudo-código. Os algoritmos **empty** (3.1), **push** (3.2), **pop** (3.3), **stackpop** (3.4) e **size** (3.5) demonstram as operações numa pilha. Estes códigos podem ser adaptados para qualquer linguagem de programação [9].

Algoritmo 3.1: Verificação se a pilha está vazia (função EMPTY(S))

```
   Input: A variável que contém a pilha (S)
   Output: Verdadeiro ou falso
1  begin
2  |  if topo de S = 0 then
3  |  |  return true
4  |  else
5  |  |  return false
6  |  endif
7  end
```

3.2 Pilhas em C

Antes de programar a solução de um problema que usa uma pilha, é necessário determinar como representar uma pilha usando as estruturas de dados existentes na linguagem de programação. Uma pilha é um conjunto ordenado de itens, e a linguagem C já contém um tipo de dado que representa um conjunto ordenado de itens: o vetor. Então, sempre que for necessário utilizar a estrutura

de pilhas para resolver um problema pode-se utilizar o vetor para armazenar esta pilha. Mas a pilha é uma estrutura dinâmica e pode crescer infinitamente, enquanto um vetor na linguagem C tem um tamanho fixo; contudo, pode-se definir este vetor com um tamanho suficientemente grande para conter esta pilha.

Algoritmo 3.2: Colocar um item na pilha (função PUSH(S,x))

```
  Input: Pilha (S) e o item a ser incluído na pilha (x)
  Output: Não tem retorno
1 begin
2 |  topo de S⟵ topo de S + 1
3 |  S(topo de S)⟵ x
4 |  return
5 end
```

Algoritmo 3.3: Retirada de um item da pilha (função POP(S))

```
  Input: Pilha (S)
  Output: Item que está no topo da pilha
1 begin
2 |  if EMPTY(S) then
3 |  |  Erro de pilha vazia
4 |  else
5 |  |  topo de S⟵ topo de S - 1
6 |  |  return S(topo de S + 1)
7 |  endif
8 end
```

Algoritmo 3.4: Pega o item do topo da pilha mas não desempilha (função STACKPOP(S))

```
  Input: Pilha (S)
  Output: Item que está no topo da pilha
1 begin
2 |  if EMPTY(S) then
3 |  |  Erro de pilha vazia
4 |  else
5 |  |  return S(topo de S)
6 |  endif
7 end
```

Algoritmo 3.5: Tamanho da pilha (função SIZE(S))

```
Input: A variável que contém a pilha (S)
Output: Quantidade de itens da pilha (topo de S)
1 begin
2 |  return topo de S
3 end
```

O programa 3.1 apresenta um exemplo de programa em C para manipulação de pilhas.

Programa 3.1: Exemplo de manipulação de pilha

```
/* programa_pilha_01.c */

#include <stdio.h>

void push(int valor);
int pop(void);
int size(void);
int stacktop(void);

int pilha[20];
int pos=0;

void push(int valor)
{
    pilha[pos]=valor;
    /* Empilha um novo elemento. Não é verificada a capacidade
       máxima da pilha.*/
    pos++;
    return;
}

int pop()
{
    /* Retorna o elemento do topo da filha. Não é verificado
       o final da pilha. */
    return (pilha[--pos]);
}

int size()
{
    return pos; /* retorna o topo da pilha */
}
```

```
33
   int stacktop() /* retorna o topo da pilha sem desempilhar */
   {
    return pilha[pos];
   }
38
   int main(int argc, char ** argv )
   {
   printf("\nColocados dados na pilha");
   push(1);
43 push(2);
   push(3);

   printf("\nTamanho da pilha %d", size());

48 printf("\nPegando dado da pilha: %d", pop());
   printf("\nPegando dado da pilha: %d", pop());
   printf("\nPegando dado da pilha: %d", pop());

   printf("\nTamanho da pilha %d", size());
53 return 0;
   }
```

Uma pilha em C pode ser declarada como uma estrutura contendo dois objetos: um vetor para armazenar os elementos da pilha e um inteiro para indicar a posição atual do topo da pilha (programa 3.2).

Programa 3.2: Exemplo de manipulação de pilha com estrutura

```
   /* programa_pilha_02.c */

   #include <stdio.h>
   #include <stdlib.h>
5  #define TAMANHO_PILHA 100

   /* Estrutura que irá conter a pilha de informações */
   struct pilha
   {
10     int topo;
       int itens[TAMANHO_PILHA];
   };

   int empty(struct pilha *p)
15 {
       if( p->topo == -1
```

```
    {
        return 1;
    }
    return 0;
}

int pop(struct pilha *p)
{
    if( empty(p) )
    {
        printf("\nPilha vazia");
        exit(1);
    }
    /* retorna o item da pilha atual e diminui a posição da pilha */
    return (p->itens[p->topo--]);
}

void push(struct pilha *p, int e)
{
    if( p->topo == (TAMANHO_PILHA - 1))
    {
        printf("\nEstouro da pilha");
        exit(1);
    }
    /* após verificar se não haveria estouro na capacidade da pilha,
       é criada uma nova posição na pilha e o elemento é armazenado */
    p->itens[++(p->topo)] = e;
    return;
}

int size(struct pilha *p)
{
    /* sempre lembrando que na linguagem C o índice de um
       vetor começa na posição 0 */
    return p->topo+1;
}

int stackpop(struct pilha *p)
{
    return p->itens[p->topo];
}

int main(void)
{
    struct pilha x;
    x.topo = -1;

    push(&x,1);
```

```
    push(&x,2);
    push(&x,3);

    printf("\nTamanho da pilha %d", size(&x));
    printf("\nElemento do topo da fila %d", stackpop(&x));

    printf("\n%d", pop(&x));
    printf("\n%d", pop(&x));
    printf("\n%d", pop(&x));
    printf("\n%d", pop(&x));
    return 0;
    }
```

3.3 Exercícios

1. Dada uma pilha P, construir uma função que inverte a ordem dos elementos dessa pilha, utilizando apenas uma estrutura auxiliar. Definir adequadamente a estrutura auxiliar e prever a possibilidade da pilha estar vazia.

2. Construir uma função que troca de lugar o elemento que está no topo da pilha com o que está na base da pilha. Usar apenas uma pilha como auxiliar.

3. Dada uma pilha contendo números inteiros quaisquer, construir uma função que coloca os pares na base da pilha e os ímpares no topo da pilha. Usar duas pilhas como auxiliares.