MINISTÈRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE ITAJUBÁ
Criada pela Lei nº 10.435 – 24/04/2002

Algoritmos e Estrutura de Dados I
Anotações de Aula
Profa Melise Maria Veiga de Paula

PILHA

Uma ***pilha*** (= *stack*) é uma lista de dados que só aceita remoção do último elemento e só aceita inserção após o último elemento. Portanto, o último a entrar é o primeiro a sair (política Last-In-First-Out).

Várias aplicações reais são implementadas usando pilhas, por exemplo, em editores de texto, as operações "undo" usam uma pilha para armazenar as últimas alterações realizadas.

A pilha é uma estrutura de dados que tem uma única extremidade, pela qual, os dados são inseridos e removidos. Esta extremidade é denominada topo da pilha.

Embora, conceitualmente, a pilha seja uma estrutura de dados dinâmica (aumenta e diminui de maneira imprevisível durante a execução do programa), basicamente, existem duas estratégias para implementar uma pilha: implementação estática e dinâmica. Na implementação estática, utilizamos um vetor para armazenar os dados na pilha. Já na implementação dinâmica, utilizamos ponteiros.

Neste momento, vamos considerar somente a implementação estática. Observem que, neste caso, temos que limitar o tamanho da pilha para que seja possível utilizar vetores.

Ao manipular uma pilha como estrutura de dados, deve-se considerar as operações que poderão ser realizadas. Neste contexto, podemos citar as seguintes operações:

- iniciaPilha
- pilhaVazia: verifica se a pilha está vazia
- pilhaCheia: verifica se a pilha está cheia
- empilha: inclui um elemento no topo da pilha (se a pilha não estiver cheia)
- desempilha: retira o elemento do topo da pilha

Além destas operações válidas, podemos considerar algumas operações para facilitar a validação das nossas operações básicas. Por exemplo, a operação top abaixo:

- top: retorna o valor do elemento do topo da pilha sem retira-lo

Vamos considerar uma aplicação que utilize uma pilha para manipular elementos inteiros. A seguir, uma possível versão para implementação dessas funções. É importante lembrar que, nossa pilha será armazenada em um vetor. Além disso, deve ser sempre possível conhecer o último elemento da pilha, denominado elemento do topo da pilha. Neste caso, o topo representa o índice do último elemento inserido no vetor (pilha).

```
#include <stdio.h>
#define MAX 100

// A pilha
int pilha[MAX], topo;

// Operações
void iniciapilha () {
  topo = -1;
}

int pilhavazia () {
  return topo == -1;
}

int pilhaCheia () {
      return topo == MAX -1;
}

void empilha (int i) {
  if (!pilhaCheia())
      pilha[++topo] = i;
  else printf("\n Pilha cheia"):
}

// Caso a pilha esteja vazia, a função desempilha deve retornar
-1
int desempilha () {
  if (!pilhaVazia())
      return pilha[topo--];
 else
      return -1;
}

// Caso a pilha esteja vazia, a função top deve retornar -1
int top () {
      if (!pilhaVazia())
       return pilha[topo];
      else
        { printf("\n Pilha vazia");
        return -1; }
}
```

Note que até agora, ainda não usamos efetivamente uma única estrutura como sendo nossa pilha, só definimos um vetor e uma variável de controle e implementamos suas operações básicas. Uma possível aplicação para esta pilha seria a conversão de decimal para binário. Neste caso, a pilha deverá ser usada para armazenar os restos das divisões por 2. Veja o código abaixo.

```
// Neste caso, as variáveis que representam a pilha foram
definidas com o escopo global
int pilha[MAX], topo;
main () {
  int num;
  printf ("\nDigite um número");
  scanf ("%d", &num);
  iniciapilha();
  while (num) {
    empilha(num%2);
    num /= 2;
  }
  printf ("\nCorrespondente Binário:");
  while (!pilhavazia())
    printf("%d ", desempilha());
}
```

Mas esta versão das funções apresenta dois inconvenientes: o uso de variáveis globais (que aumenta muito o risco de erros quando do uso das variáveis) e o fato da pilha não ter sido definida como uma única estrutura. Repare que, nesta versão, o "local" da pilha onde os dados são armazenados e o topo da pilha foram definidos de forma independente. Uma forma de resolver isso seria criar um tipo definido usando uma estrutura.

```
//Nossa segunda versão da pilha

typedef struct {
        int dado[MAX];
        int topo;
}tpilha;

tpilha p;

//Nossas funções agora mudariam pouco. Analise o código abaixo:

void iniciapilha (void) {
  p.topo = -1;
}

int pilhavazia (void) {
  return p.topo == -1;
}

int pilhaCheia (void) {
     return p.topo == MAX -1;
}

void empilha (int i) {
  if (!pilhaCheia())
      p.dado[++p.topo] = i;
```

```
  else printf("\n Pilha cheia"):
}

// Caso a pilha esteja vazia, a função desempilha deve retornar
-1
int desempilha (void) {
  if (!pilhaVazia())
      return p.dado[p.topo--];
else
      return -1;
}

// Caso a pilha esteja vazia, a função top deve retornar -1
int top () {
      if (!pilhaVazia())
       return p.dado[p.topo];
      else
        { printf("\n Pilha vazia");
        return -1; }
}
```

Nossa pilha começa a ficar melhor, mas ainda temos um problema: ainda usamos variáveis globais para representar nossa pilha. Mas isso pode ser resolvido, nossa pilha pode ser declarada dentro da função main. Para isso, precisamos definir alguma estratégia para que as alterações da pilha realizadas dentro das operações sejam permanentes, ou seja, permaneçam mesmo fora do escopo das respectivas funções. Para isso, basta passar nossa pilha como referência. Portanto, em todas as funções que alteram a estrutura da nossa pilha (seja o topo ou os dados), vamos ter que passar um ponteiro para a pilha. Nas demais, ou seja, nas funções que usam a pilha, mas que não alteram a estrutura da pilha, podemos passar a pilha como valor.

Para aplicação da conversão de binários, nosso código ficaria assim:

```
#include<stdio.h>
#include<stdlib.h>
#define MAX 100

typedef struct {
      int dado[MAX];
      int topo;
} pilha;

void inicializaPilha (pilha *p){
   p->topo = -1;
}

int pilhaVazia (pilha p) {
   return p.topo==-1;
}
```

```
int pilhaCheia (pilha p) {
    return p.topo == MAX-1;
}

void empilha (pilha *p, int i) {
    if (pilhaCheia(*p)) {
         printf("\nPilha cheia");
    }
    else {
        p->topo++;
        p->dado[p->topo] = i;
    }
}

int desempilha (pilha *p) {
   if (!pilhaVazia(*p)) {
         return p->dado[p->topo--];
    }
    else return -1;
}

int main (){
   int num, i;
   pilha p;
   printf("\nDigite um num: ");
   scanf("%d",&num);
   inicializaPilha(&p);
   while (num >= 1) {
        empilha(&p,num%2);
        num = num/2;
   }
   printf("Binario: \n");
   while (!pilhaVazia(p))
      printf(" %d ", desempilha(&p));
   printf(" \n ");
   system("pause");
}
```