Algoritmos e Estrutura de Dados

Pilhas

Prof. Nilton Luiz Queiroz Jr.

Pilhas

- Um tipo abstrato de dado onde os dados são inseridos “um sobre o outro”;
- Podem ser aplicadas em:
  - Chamadas de funções aninhadas;
  - Recursividade;
  - Verificar a corretude de parênteses;
  - Delimitadores de blocos ({ e } em C);

Pilhas

- Alguns modelos intuitivos de pilha:
  - Livros colocados um sobre outro;
  - Pratos colocados um sobre outro;

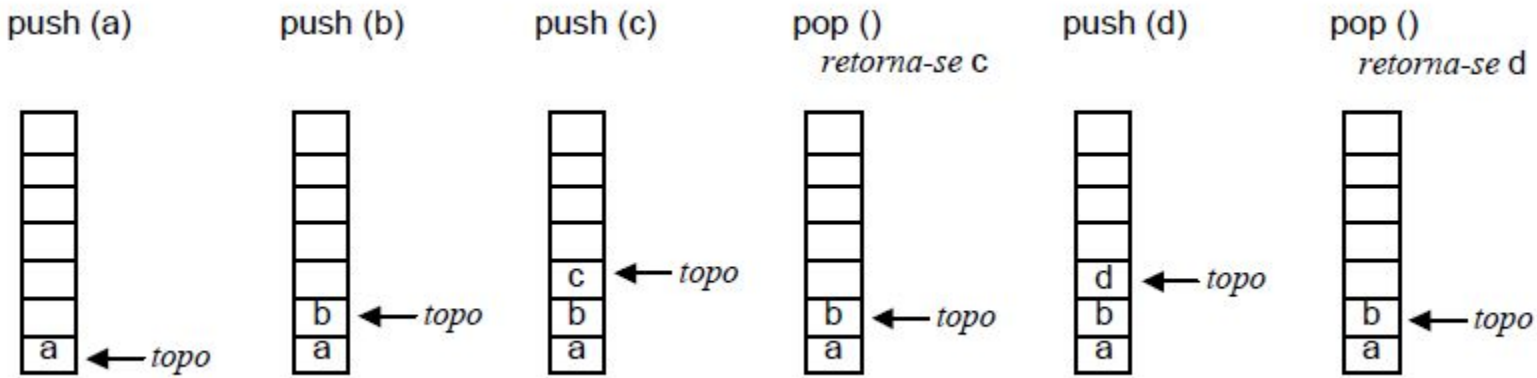
Pilhas

- Obedece uma política FILO (first in last out)
  - O primeiro elemento a entrar é o último a sair;
- A idéia fundamental de uma pilha é:
  - Todo acesso aos elementos é feito pelo topo:
    - Ao introduzir um novo elemento na pilha ele é colocado no topo;
    - Ao remover um elemento da pilha ele é retirado no topo;
- Como a remoção só é feita no topo, é comum a possibilidade de inserção de elementos repetidos;

Pilhas

- Toda pilha tem duas operações básicas:
  - Push (Empilhar);
    - Inserir um novo elemento no topo da pilha;
  - Pop (Desempilhar);
    - Remover o elemento do topo da pilha;

Pilhas


push (a)
a
topo
push (b)
b
a
topo
push (c)
c
b
a
topo
pop ()
retorna-se c
b
a
topo
push (d)
d
b
a
topo
pop ()
retorna-se d
b
a
topo

Pilhas

- Dentre as maneiras de se implementar pilhas temos:
  - Estática;
    - É implementada em um vetor;
    - Quantidade de memória alocada limita o crescimento da pilha;
  - Dinâmica;
    - É implementada em lista;
    - Não se fica preso ao tamanho máximo alocado para pilha;
      - Quantidade de memória alocada cresce e diminui conforme elementos são inseridos e removidos;

Pilhas estáticas

- Pilhas estáticas são implementadas em vetor;
- Um possível conjunto de operações sobre pilhas estáticas é composto pelas operações de:
  - Inicializar;
  - Verificar se a pilha está vazia;
  - Verificar se a pilha está cheia;
  - Empilhar;
  - Desempilhar;
  - Imprimir os valores da pilha;

Pilhas estáticas

- Para implementação das pilhas estáticas precisamos dos seguintes agregados heterogêneos:
  - O tipo do item que será armazenado;
  - O tipo que contém as informações da pilha;

```
struct tipo_item{
    int chave;
};

struct tipo_pilha{
    int topo;
    struct tipo_item dados[TAM];
};
```

Pilhas estáticas

- Inicializar pilha:

```
void inicializa(struct tipo_pilha *p){
    p->topo=0;
}
```

- Verificar se a pilha está vazia:

```
int cheia(struct tipo_pilha *p){
    return p->topo==TAM;
}
```

- Verificar se a pilha está vazia:

```
void inicializa(struct tipo_pilha *p){
    p->topo=0;
}
```

Pilha estática

- Empilhar;
  - Não se pode empilhar em uma pilha cheia;

```
int empilha(struct tipo_pilha *p, struct tipo_item x){
    if(!cheia(p)){
        /*codigo para empilhar*/
        return 1;
    }else{
         return 0;
    }
}
```

Pilha estática

- Empilhar;
  - Não se pode empilhar em uma pilha cheia;

```
int empilha(struct tipo_pilha *p, struct tipo_item x){
    if(!cheia(p)){
        p->dados[p->topo]=x;
        p->topo++;
        return 1;
    }else{
        return 0;
    }
}
```

Pilha estática

- Desempilhar
  - Não se pode desempilhar de uma pilha vazia

```
int desempilha(struct tipo_pilha *p, struct tipo_item *x){
    if(!vazia (p)){
        /*código para desempilhar*/
         return 1;
    }else{
         return 0;
    }
}
```

Pilha estática

- Desempilhar
  - Não se pode desempilhar de uma pilha vazia

```
int desempilha(struct tipo_pilha *p, struct tipo_item *x){
    if(!vazia (p)){
        p->topo--;
        *x=p->dados[p->topo];
        return 1;
    }else{
        return 0;
    }
}
```

Pilhas dinâmicas

- Pilhas dinâmicas são implementadas em listas;
- Devemos implementar operações que façam o seguinte:
  - Inicializar a pilha;
  - Verificar se a pilha está vazia;
  - Empilhar;
  - Desempilhar;
  - Obter elemento do topo;
- Além disso é necessário as estruturas de dados para a pilha;

Implementação de pilha

- Estruturas para uma pilha dinâmica:

```
struct tipo_item{
    int chave;
};

struct tipo_celula{
    struct tipo_celula *prox;
    struct tipo_item item;
};

struct tipo_pilha{
    struct tipo_celula *topo;
};
```

Implementação de pilha

- Inicializar

```
void inicializa(struct tipo_pilha *p){
    p->topo=(struct tipo_celula*)malloc(sizeof(struct tipo_celula));
    p->topo->prox=NULL;
}
```

Implementação de pilha

- Verificar se a pilha está vazia

```
int vazia(struct tipo_pilha *p){
    return p->topo->prox == NULL;
}
```

Implementação de pilha

- Empilhar

```
void empilha(struct tipo_pilha *p, struct tipo_item x){
    struct tipo_celula *novo;
    /*algoritmo*/
}
```

Implementação de pilha

- Empilhar

```
void empilha(struct tipo_pilha *p, struct tipo_item x){
    struct tipo_celula *novo;
    novo=(struct tipo_celula*)malloc(sizeof(struct tipo_celula));
    novo->prox=p->topo->prox;
    novo->item=x;
    p->topo->prox=novo;
}
```

Implementação de pilha

- Desempilhar

```
int desempilha(struct tipo_pilha *p, struct tipo_item *x){
    struct tipo_celula *del;
    if(vazia(p)){
        return 0;
    }else{
        del=p->topo->prox;
        *x=del->item;
        p->topo->prox=del->prox;
        free(del);
        return 1;
    }
}
```

Implementação de pilha

- Obter elemento do topo

```
int topo(struct tipo_pilha *p, struct tipo_item *x){
   /*algoritmo*/
}
```

Obs: O valor do topo estará contido na segunda variável passada como parâmetro para a função. Assim, antes de usá-lo é importante verificar se a função retornou o valor que indica que a operação foi bem sucedida.

Implementação de pilha

- Obter elemento do topo

```
int topo(struct tipo_pilha *p, struct tipo_item *x){
    if(vazia(p)){
        return 0;
    }else{
        *x=p->topo->prox->item;
        return 1;
    }
}
```

Obs: O valor do topo estará contido na segunda variável passada como parâmetro para a função. Assim, antes de usá-lo é importante verificar se a função retornou o valor que indica que a operação foi bem sucedida.

Exercícios

1. Faça um subprograma que receba uma string contendo uma expressão matemática e diga se os parênteses da expressão estão corretos (ou seja, toda vez que um parêntese é aberto um outro é fechado). Não é necessário verificar o resto da expressão.
2. Faça um subprograma que copie uma pilha para outra. Deve-se usar somente estruturas auxiliares e operações de pilha. Esse subprograma deve funcionar tanto para pilhas estáticas quanto para dinâmicas.
3. Utilizando uma pilha faça um subprograma que receba um número decimal e escreva esse número em binário.
4. Implemente a operação esvaziar pilha para:
    a. Uma pilha estática;
    b. Uma pilha dinâmica;

Referências

ZIVIANI, N.; Projeto de Algoritmos com Implementações em Pascal e C, 3ª Edição, Livraria Pioneira Editora, 1996.

TANENBAUM, A. M., Langsam, Y., Augenstein, M.- Estruturas de Dados Usando C. Editora Makron Books, 1995.