Universidade do Mindelo
Departamento de Engenharia e Recursos do Mar
Licenciatura em Engenharia Informática e Sistemas Computacionais

Programação em Sistemas Computacionais

Ano Lectivo de 2022/2023 — 2 de Junho de 2023
Trabalho Prático 2 — Implementar uma Stack com Lista Ligada

Estrutura de Dados Stack

Stack é uma estrutura de dados que pode ser usada para armazenar conjunto de dados de um programa. Ao contrário do *array* que permite aceder à qualquer elemento (a[4]), numa Stack somente o último elemento que foi adicionado á estrutura pode ser acedida. De igual forma, somente podemos adicionar valores no topo da Stack, sendo que as operações realizadas só afectam dados que estão no topo. Na representação da estrutura de dados Stack os valores podem ser adicionados usando um comando **push** e serão removidos usando o comando **pop**.

Como exemplo, se temos uma Stack **A** que neste caso é uma estrutura que armazena valores inteiros. Para adicionar um valor seria usar os comandos:

- push(A,5)
- push(A,2)
- push(A,6)

No nosso exemplo a Stack A fica com o seguinte conteúdo:

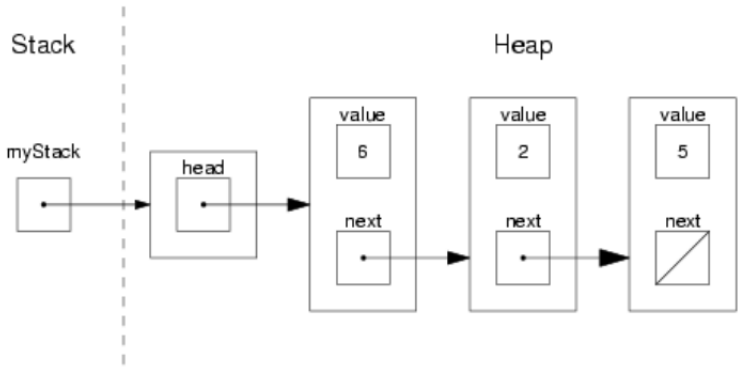
- 6
- 2
- 5

O comando pop retornará o elemento no topo da Stack. No nosso exemplo o valor 6 em primeiro lugar. Depois o valor 2 e á seguir o 5.

Tem continuação...

Implementação de Stack: Lista ligada

1. Uma possível implementação para a estrutura Stack pode ser realizada com lista ligada, conceito que já foi objecto de estudo nesta disciplina e será adoptado neste trabalho. Como exemplo, considera uma representação da uma lista ligada com os três valores na Stack apresentada anteriormente, tal como a figura que se segue:



Na fígura, a primeira caixa mais à esquerda, representa um ponteiro de tipo **stackT *** localizada na área de memória estática. Este ponteiro, guarda o endereço de uma estrutura stackT na área de memória dinâmica, sendo esta área que irá conter os nós da lista ligada. Ou seja, **stackT** é uma struct C que tem um campo do tipo ponteiro para o início da **Stack**, designado de *head* sendo o tipo stackT definido pela seguinte struct C

```
typedef struct {
    nodeT *head;
} stackT;
```

Como se pode ver, é declarado o ponteiro *head* para uma estrutura do tipo **nodeT**, sendo que este tipo é que define um nó na lista ligada. O referido nó é um tipo especificado da seguinte forma:

```
typedef struct _nodeT {
    valueT value;
    struct _nodeT *next;
} nodeT;
```

No nosso trabalho o tipo **valueT** representa dados de processos num sistema operativo e para já iremos manter somente identificador do processo e nome do comando ou programa que deu origem ao processo.

 Tem continuação...

Trabalho a realizar

1. Definir a struct C valueT que representa um processo como referido anteriormente.

2. Usar as struct atrás referidas e implementar uma série de funções para manipular os dados na Stack. As funções que vai implementar são as seguintes:

   - nodeT *NewNode(void);
   - stackT *NewStack(void);
   - void Push(stackT *stack, valueT processo);
   - valueT Pop(stackT *stack);
   - void EmptyStack(stackT *stack);
   - void FreeStack(stackT *stack);
   - int isEmpty(stackT *stack);

   As funções têm as descrições apresentadas a seguir:

   1. NewStack - Função retorna um ponteiro para um tipo stackT. O ponteiro head será inicializado com NULL. Ao ser invocado esta função, na forma myStack = NewStack(), myStack será um ponteiro do tipo stackT * e guarda o endereço de uma nova estrutura stackT. Esta, representa uma nova Stack criada com a função. Deve ser apresentada mensagem de erro e retornar NULL caso não haja mais espaço disponível na heap para criar nova Stack.
   2. Push - Função deve receber como argumento um ponteiro para um stackT e um *processo* a ser adicionado na Stack. Com o push de um processo na stack deverá ser feito o seguinte:
      - alocar espaço para um tipo nodeT no heap com uma função NewNode
      - fazer o campo next do novo nó apontar para o nó head da Stack
      - fazer o campo head da Stack apontar para o novo nó
   3. Pop - Função recebe como argumento um ponteiro para estrutura stackT e deve retornar o processo que está no topo. Por conseguinte deverá:
      - Retornar 0 e mostrar uma mensagem de erro se a Stack estiver vazia.
      - Colocar o campo next a apontar para o próximo nó
      - Libeertar o espaço que era ocupado pelo nó anterior.
   4. EmptyStack - Esta função recebe como argumento um ponteiro para uma Stack e deve libertar o espaço alocado para cada nó na lista ligada. Porém, não deverá ser libertado o espaço alocado para estrutura stackT. Deverá sim atribuir NULL ao ponteiro head. Por outras palavras não se pretende destruir a Stack, somente, esvaziar [remover todos os processos]
   5. FreeStack - Esta função recebe como argumento um ponteiro para uma Stack e deve libertar o espaço alocado para estrutura stackT. A função deverá libertar espaço somente para uma Stack que não contem processos ou seja vazia. Caso seja utilizada numa Stack que contem dados deve ser apresentada uma mensagem de erro.
   6. isEmpty - Esta função recebe como argumento um ponteiro para uma Stack e retornará 1 se a Stack está vazia ou 0 senão. Uma Stack diz-se vazia se o ponteiro head é NULL

 Tem continuação...

7. NewNode - A função deverá retornar um ponteiro para nodeT na heap ou NULL e escrever uma mensagem de erro caso não houver espaço para fazer isso.

Entrega

1. Programa implementado na Linguagem C contendo todo o código utilizado e um relatório técnico explicando a implementação realizada
2. Data de entrega : 9 de Junho de 2023 até as 18:59
3. Entrega no classroom