Tipos abstratos de dados; pilhas e filas

Além do Sedgewick (**sempre** leiam o Sedgewick), veja

- `http://www.ime.usp.br/~pf/algoritmos/aulas/pilha.html`
- `http://www.ime.usp.br/~pf/algoritmos/aulas/fila.html`
- `http://www.ime.usp.br/~pf/mac0122-2002/aulas/footnotes/abstract-data-type.html`

Tipos abstratos de dados

Definição 1. *Um tipo de dados é um conjunto de valores munido de um conjunto de operações.*

- Exemplos: `int`, `double` e `char`

Tipos abstratos de dados

Definição 2. *Um tipo abstrato de dados (TAD, ou ADT em inglês) é um tipo de dados definido através de seu comportamento, sem qualquer referência a sua implementação. Acesso a uma ADT é feito unicamente através de uma interface.*

- Em C, a interface é implementada através de arquivos `.h`
- Também falamos de *clientes* e *implementações*
- Exemplos: filas generalizadas, pilhas, filas e tabelas de símbolos

Tipos abstratos de dados

- Filas generalizadas: estruturas que armazenam objetos de um dado tipo, permitindo as operações básicas de

  - inserção
  - remoção

Outras operações típicas: inicialização, teste para verificar se a estrutura está vazia, dentre outras

Objetos abstratos

▷ Em nossas pilhas, filas, tabelas de símbolos, algoritmos de ordenação, manipularemos objetos do tipo `Item`.

Podemos implementar `Item` através de arquivos `.h` como seguem:

```
/* Item.h */
typedef int Item
#define eq(A, B) (A == B)
```

```
/* Item.h */
typedef char* Item
#define eq(A, B) ((strcmp(A,B)) == 0)
#define less(A, B) ((strcmp(A,B)) < 0)
```

Objetos abstratos

▷ Podemos implementar `Item` com um par `Item.h`/`Item.c` (possivelmente tornando-o uma ADT).

▷ Veremos mais exemplos futuramente.

Pilhas e filas

Definição 3. *Uma pilha é uma ADT que admite duas operações: empilhar (correspondente à inserção) e desempilhar, que remove o item mais recentemente empilhado.*

▷ Política *last-in-first-out*; empilhar = *push*, desempilhar = *pop*

Definição 4. *Uma fila é uma ADT que admite duas operações: enfileirar (correspondente à inserção) e retirar, que remove o item inserido na fila há mais tempo.*

▷ Política *first-in-first-out*; enfileirar = *put*, retirar = *get*

Pilhas: interface

```
/* STACK.h */
void STACKinit(int);
 int STACKempty();
void STACKpush(Item);
Item STACKpop();
```

Pilhas: um cliente

```
/* prog4.2.c */
#include <stdio.h>
#include <string.h>
#include "Item.h"
#include "STACK.h"

int main(int argc, char *argv[])
  { char *a = argv[1]; int i, N = strlen(a);
    STACKinit(N);
```

Pilhas: um cliente

```
/* prog4.2.c */
[...]
    for (i = 0; i < N; i++)
      {
        if (a[i] == '+')
          STACKpush(STACKpop()+STACKpop());
        if (a[i] == '*')
          STACKpush(STACKpop()*STACKpop());
        if ((a[i] >= '0') && (a[i] <= '9'))
          STACKpush(0);
        while ((a[i] >= '0') && (a[i] <= '9'))
          STACKpush(10*STACKpop() + (a[i++]-'0'));
      }
```

Pilhas: um cliente

```
/* prog4.2.c */
[...]
    printf("%d \n", STACKpop());
    return 0;
  }
```

Pilhas: uma implementação (com vetores)

```
/* prog4.4.c */
#include <stdlib.h>
#include "Item.h"
#include "STACK.h"
static Item *s;
static int N;
void STACKinit(int maxN)
  { s = malloc(maxN*sizeof(Item)); N = 0; }
int STACKempty()
  { return N == 0; }
void STACKpush(Item item)
  { s[N++] = item; }
Item STACKpop()
  { return s[--N]; }
```

Pilhas: uma implementação (com listas ligadas)

```
/* prog4.5.c */
#include <stdlib.h>
#include "Item.h"

typedef struct STACKnode* link;
struct STACKnode { Item item; link next; };

static link head;

link NEW(Item item, link next)
  { link x = malloc(sizeof *x);
    x->item = item; x->next = next;
    return x;
  }
```

Pilhas: uma implementação (com listas ligadas)

```
/* prog4.5.c */
[...]
void STACKinit(int maxN)
  { head = NULL; }

int STACKempty()
  { return head == NULL; }

void STACKpush(Item item)
  { head = NEW(item, head); }
```

Pilhas: uma implementação (com listas ligadas)

```
/* prog4.5.c */
[...]
Item STACKpop()
  { Item item = head->item;
    link t = head->next;
    free(head); head = t;
    return item;
  }
```

Exemplo de execução

```
yoshi@erdos:~/Main/www/2006ii/mac122a/exx$ cat Item.h
typedef int Item;
yoshi@erdos:~/Main/www/2006ii/mac122a/exx$ make prog4.2.o prog4.4.o
gcc -g -I. -Wall -pedantic -ansi -c prog4.2.c
gcc -g -I. -Wall -pedantic -ansi -c prog4.4.c
yoshi@erdos:~/Main/www/2006ii/mac122a/exx$ gcc prog4.2.o prog4.4.o
yoshi@erdos:~/Main/www/2006ii/mac122a/exx$ a.out "2 3 +"
5
yoshi@erdos:~/Main/www/2006ii/mac122a/exx$
```

Exemplo de execução

```
yoshi@erdos:~/Main/www/2006ii/mac122a/exx$ a.out "59 8 + 4 61 * + 7 + 2 *"
636
yoshi@erdos:~/Main/www/2006ii/mac122a/exx$ make prog4.5.o
gcc -g -I. -Wall -pedantic -ansi -c prog4.5.c
yoshi@erdos:~/Main/www/2006ii/mac122a/exx$ gcc prog4.2.o prog4.5.o
yoshi@erdos:~/Main/www/2006ii/mac122a/exx$ a.out "59 8 + 4 61 * + 7 + 2 *"
636
yoshi@erdos:~/Main/www/2006ii/mac122a/exx$
```

Exercício

Exercício 5. *Escreva um programa que use uma pilha para inverter a ordem das letras de cada palavra de um string, preservando a ordem das palavras. Por exemplo, dado o texto* ESTE EXERCICIO E MUITO FACIL *a saída deve ser* ETSE OICICREXE E OTIUM LICAF*. (Lembre-se: strings em C terminam com* `'\0'`*.)*

Exercício

Exercício 6. *Digamos que nosso alfabeto é formado pelas letras* a, b *e* c. *Considere o seguinte conjunto de cadeias de caracteres sobre nosso alfabeto:*

$$\mathtt{c}, \mathtt{aca}, \mathtt{bcb}, \mathtt{abcba}, \mathtt{bacab}, \mathtt{aacaa}, \mathtt{bbcbb}, \ldots$$

Qualquer cadeia deste conjunto tem a forma $W\mathtt{c}M$*, onde* W *é uma seqüência de letras que só contém* a *e* b *e* M *é o inverso de* W*, ou seja,* M *é* W *lido de trás para frente. Escreva um programa que determina se uma cadeia* X *pertence ou não ao nosso conjunto, ou seja, determina se* X *é da forma* $W\mathtt{c}M$.

Exercício

Exercício 7. *Escreva uma função de protótipo*

```
char *intopost(char *inf)
```

que recebe como entrada um string `inf` *que contém uma expressão aritmética em notação infixa e que devolve a expressão posfixa correspondente. Você pode supor que os operandos em* `inf` *são dígitos.* Sugestão: *Use uma pilha de caracteres.*

▷ Exemplo: `(1 + ( 2 * 3 ) )` deve resultar em `1 2 3 * +`;
`( ( 1 - ( 2 + 3 ) ) * (4 + 5) )` deve resultar em
`1 2 3 + - 4 5 + * .`

Exercício

- Para fazer o exercício anterior, você pode se inspirar em

`http://www.ime.usp.br/~yoshi/2006ii/mac122a/exx/prog4.3.c`

Exercício

Exercício 8. *A implementação de pilha que vimos trata do caso de termos uma única pilha em nosso programa. Como seria um bom esquema para termos mais de uma pilha?*