



CENTRO DE CIÊNCIA E TECNOLOGIA
LABORATÓRIO DE CIÊNCIAS MATEMÁTICAS
UNIVERSIDADE ESTADUAL DO NORTE FLUMINENSE

Aplicações de Pilhas

Parte 1

Disciplina: Estrutura de Dados I

Prof. Fermín Alfredo Tang Montané

Curso: Ciência da Computação

Aplicações de Pilhas (Stacks)

- Estudaremos quatro grupos de aplicações de pilhas:
 - Inversão de dados;
 - Análise sintática (*Parsing*);
 - Adiamiento do uso de dados;
 - *Backtracking*.

Aplicações de Pilhas (Stacks)

- Inversão de dados;
 - Inversão de uma série numérica;
 - Conversão de um decimal em Binário.
- Análise sintática (*Parsing*)
 - Casamento de parênteses;

Aplicações de Pilhas (Stacks)

Inversão de dados

P3-15.c

```
1  /* This program reverses a list of integers read
2     from the keyboard by pushing them into a stack
3     and retrieving them one by one.
4     Written by:
5     Date:
6  */
7  #include <stdio.h>
8  #include <stdbool.h>
9  #include "stacksADT.h"
10
11 int main (void)
12 {
13     // Local Definitions
14     bool done = false;
15     int* dataPtr;
16
17     STACK* stack;
18
19     // Statements
20     // Create a stack and allocate memory for data
21     stack = createStack ();
22
23     // Fill stack
24     while (!done)
25     {
```

Esta aplicação usa uma pilha pra inverter uma lista de inteiros ingressados do teclado.

// Define ponteiro ao cabeçalho da pilha

// Cria a pilha

Aplicações de Pilhas (Stacks)

Inversão de dados

P3-15.c (Continuação...)

```
26     dataPtr = (int*) malloc (sizeof(int));
27     printf ("Enter a number: <EOF> to stop: ");
28     if ((scanf ("%d" , dataPtr)) == EOF
29         || fullstack (stack))
30         done = true;
31     else
32         pushStack (stack, dataPtr);
33 } // while
34
35 // Now print numbers in reverse
36 printf ("\n\nThe list of numbers reversed:\n");
37 while (!emptyStack (stack))
38 {
39     dataPtr = (int*)popStack (stack);
40     printf ("%3d\n", *dataPtr);
41     free (dataPtr);
42 } // while
43
44 // Destroying Stack
45 destroyStack (stack);
46 return 0;
47 } // main
```

// Aloca memória ao dado

// Leitura do dado

// Insere o dado na pilha

// Remove o dado na pilha

Aplicações de Pilhas (Stacks)

Inversão de dados

Results:

```
Enter a number: <EOF> to stop: 3
Enter a number: <EOF> to stop: 5
Enter a number: <EOF> to stop: 7
Enter a number: <EOF> to stop: 16
Enter a number: <EOF> to stop: 91
Enter a number: <EOF> to stop:
```

The list of numbers reversed:

```
91
16
7
5
3
```

O resultado da execução da inversão do dado.

obs. para EOF tente:

ctrl+Z ou ctrl+D

Aplicações de Pilhas (Stacks)

Inversão de dados – Exemplo

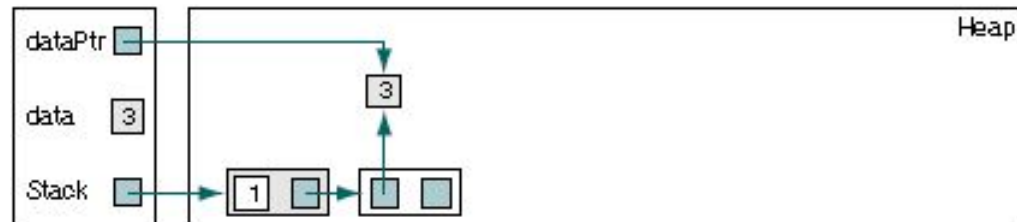
Estruturas criadas durante a execução.



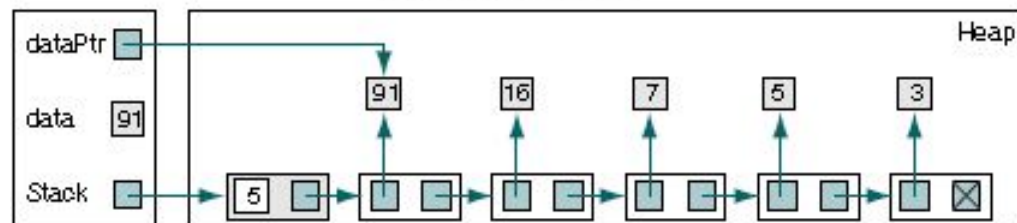
(a) Before stack creation



(b) After stack creation



(c) After first insertion

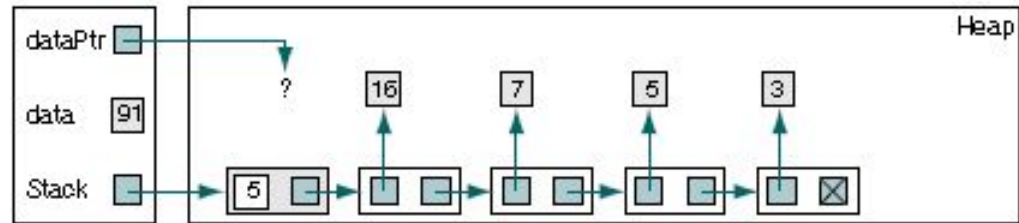


(d) After last insertion

Aplicações de Pilhas (Stacks)

Inversão de dados – Exemplo

Estruturas criadas durante a execução.



(e) After first print



(f) After last print



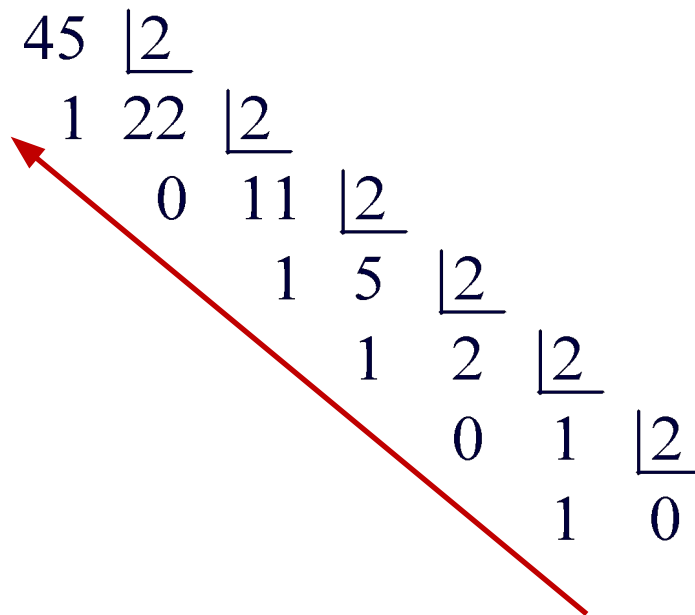
(g) After stack destruction

Aplicações de Pilhas (Stacks)

Conversão de decimal a Binário - Algoritmo

```
1 read (number)
2 loop (number > 0)
  1 set digit to number modulo 2
  2 print (digit)
  3 set number to quotient of number / 2
3 end loop
```

```
// digit <- number % 2
// colocar em uma pilha
// number <- number / 2
```



$$(45)_{10} = (101101)_2$$

A figura ilustra o algoritmo para conversão de um inteiro em binário, que requer o uso de uma pilha, pois os dígitos gerados precisam ser invertidos.

Aplicações de Pilhas (Stacks)

Conversão de decimal a Binário

P3-16.c

```
1  /* This program reads an integer from the keyboard
2     and prints its binary equivalent. It uses a stack
3     to reverse the order of 0s and 1s produced.
4     Written by:
5     Date:
6  */
7  #include <stdio.h>
8  #include "stacksADT.h"
9
10 int main (void)
11 {
12     // Local Definitions
13     unsigned int    num;
14     int*            digit;
15     STACK* stack;
16
17     // Statements
18     // Create Stack
19     stack = createStack ();
20
21     // prompt and read a number
22     printf ("Enter an integer: ");
23     scanf ("%d", &num);
24 }
```

Esta aplicação implementa o algoritmo de conversão decimal para binário. Usa uma pilha para armazenar os dígitos gerados pela operação modulo e assim poder inverter a sequência.

// Define ponteiro ao cabeçalho da pilha

// Cria a pilha

// Ingressa um número decimal

Aplicações de Pilhas (Stacks)

Conversão de decimal a Binário

P3-16.c (Continuação...)

```
25 // create 0s and 1s and push them into the stack
26 while (num > 0)
27 {
28     digit = (int*) malloc (sizeof(int));
29     *digit = num % 2;
30     pushStack (stack, digit);
31     num = num / 2;
32 } // while
33
34 // Binary number created. Now print it
35 printf ("The binary number is : ");
36 while (!emptyStack (stack))
37 {
38     digit = (int*)popStack (stack);
39     printf ("%1d", *digit);
40 } // while
41 printf ("\n");
42
43 // Destroying stack
44 destroyStack (stack);
45 return 0;
46 } // main
```

// Aloca memória ao dígito
// Calcula o dígito
// Insere o dígito na pilha

// Remove o dígito da pilha
// Imprime o dígito

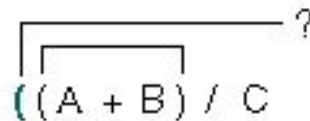
Results:

Enter an integer: 45
The binary number is : 101101

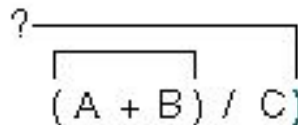
Aplicações de Pilhas (Stacks)

Análise sintática (Parsing)

- Analisar o código de um programa para determinar se todos os parênteses estão corretamente casados (emparelhados).


`((A + B) / C`

(a) Opening parenthesis not matched


`(A + B) / C)`

(b) Closing parenthesis not matched

- Existem duas situações de erro possíveis:
 - i) Ter parênteses de abertura a mais. Neste caso, após o processamento, a pilha deveria ficar vazia, mas ficará com parênteses de abertura.
 - ii) Ter parênteses de fechamento a mais. Neste caso, a pilha ficará vazia durante o processamento, revelando o excesso de parênteses de fechamento .

Aplicações de Pilhas (Stacks)

Análise sintática (Parsing) - Algoritmo

- O algoritmo utiliza uma pilha para armazenar parênteses de abertura e comparar em ordem inversa com os parênteses de fechamento.

```
Algorithm parseParens
This algorithm reads a source program and parses it to make
sure all opening-closing parentheses are paired.
1 loop (more data)
  1 read (character)
  2 if (opening parenthesis)
    1 pushStack (stack, character)
  3 else
    1 if (closing parenthesis)
      1 if (emptyStack (stack))
        1 print (Error: closing parenthesis not matched)
      2 else
        1 popStack(stack)
      3 end if
    2 end if
  4 end if
2 end loop
3 if (not emptyStack (stack))
  1 print (Error: opening parenthesis not matched)
end parseParens
```

Inserir na Pilha

Erro 2

Remove da Pilha

Erro 1

Aplicações de Pilhas (Stacks)

Análise sintática (Parsing)

P3-17.c

```
1  /* This program reads a source program and parses it to
2     make sure all opening-closing parentheses are paired.
3     Written by:
4     Date:
5  */
6  #include <stdio.h>
7  #include "stacksADT.h"
8
9  // Error Messages
10 const char closMiss[] = "Close paren missing at line";
11 const char openMiss[] = "Open paren missing at line";
12
13 int main (void)
14 {
15     // Local Definitions
16     STACK* stack;
17     char    token;
18     char*   dataPtr;
19     char    fileID[25];
20     FILE*   fpIn;
21     int     lineCount = 1;
22 }
```

Esta aplicação lê um arquivo de texto que corresponde ao um programa e verifica se existe casamento entre os parênteses de abertura e fechamento.

Aplicações de Pilhas (Stacks)

Análise sintática (Parsing)

P3-17.c (Continuação...)

```
23 // Statements
24 // Create Stack
25 stack = createStack ();
26 printf ("Enter file ID for file to be parsed: ");
27 scanf ("%s", fileID);
28
29 fpIn = fopen (fileID, "r");
30 if (!fpIn)
31     printf("Error opening %s\n", fileID), exit(100);
32
```

// Cria a pilha

// Ingressa o nome do arquivo:
// Tentar: i) close-miss;
// ii) no-error; // iii) open-miss.

// Abre arquivo

Aplicações de Pilhas (Stacks)

Análise sintática (Parsing)

P3-17.c (Continuação...)

```
33 // read characters from the source code and parse
34 while ((token = fgetc (fpIn)) != EOF )
35 {
36     if (token == '\n')
37         lineCount++;
38     if (token == '(' )
39     {
40         dataPtr = (char*) malloc (sizeof (char));
41         pushStack (stack, dataPtr);
42     } // if
43     else
44     {
45         if (token == ')')
46         {
47             if (emptyStack (stack))
48             {
49                 printf ("%s %d\n",
50                     openMiss, lineCount);
51                 return 1;
52             } // if true
53             else
54                 popStack (stack);
55             } // token ==
56         } // else
57     } // while
```

Obtém caracter

Adicionar!!

*dataPtr = token;

Insere na Pilha

Erro 2

Remove da Pilha

Aplicações de Pilhas (Stacks)

Análise sintática (Parsing)

P3-17.c (Continuação...)

```
58
59     if (!emptyStack (stack))
60     {
61         printf ("%s %d\n", closMiss, lineCount);
62         return 1;
63     } // if
64
65     // Now destroy the stack
66     destroyStack (stack);
67     printf ("Parsing is OK: %d Lines parsed.\n",
68           lineCount);
69     return 0;
70 } // main
```

Erro 1

Sem erros!!

Results:

Run 1:

Enter file ID for file to be parsed: no-errors.txt
Parsing is OK: 65 Lines parsed.

Run 2:

Enter file ID for file to be parsed: close-match.txt
close paren missing at line 46

Run 3:

Enter file ID for file to be parsed: open-match.txt
open paren missing at line 23

Referências

- Gilberg, R.F. e Forouzan, B. A. Data Structures_A Pseudocode Approach with C. Capítulo 3. Stacks. Segunda Edição. Editora Cengage, Thomson Learning, 2005.