

ESTRUTURA DE DADOS

A ESTRUTURA DE DADOS – PILHA

Olá!

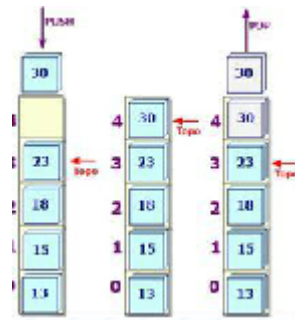
Ao final desta aula, você será capaz de:

1. Conceituar a Estrutura de Dados Pilha;
2. Representar a Estrutura de Dados Pilha por contiguidade;
3. Compreender e implementar as operações com Pilhas;
4. Compreender e testar aplicação com Pilha sequencial.

1 Empilhar para organizar

"Uma pilha é um tipo especial de Lista Linear em que todas as operações de inserção e remoção são realizadas numa mesma extremidade, denominada topo." Nesta aula, estudaremos a Pilha uma das mais simples Estruturas de Dados, mas com muitas aplicações. Lendo a definição acima, podemos concluir que a remoção do elemento acontece num processo inverso ao da inserção, isto é, o último a entrar é o primeiro a sair. Este critério que determina a sequência de entrada e saída desta forma, chama-se LIFO(Last In First Out).

Observe a figura abaixo.



Você percebe que o número 30 foi o último a entrar na Pilha e passa a se posicionar no topo. Na remoção, o número 30 é o primeiro a sair.

Uma das características da Pilha é que a inserção, remoção e acesso acontecem somente em uma extremidade como você pode observar na figura. Normalmente, apresento o vetor como uma matriz linha, mas para deixar mais parecido com uma Pilha, apresentei-o como matriz coluna e ainda dei um giro de 180°. Tudo pelo "bem da Pilha".

Sei que você deve estar curioso querendo saber por que o 30 da última Pilha ficou cinza e não foi removido. Não é verdade? Quando estivermos construindo o trecho de remoção, você entenderá.

Os procedimentos de inserção e remoção lembram qualquer empilhar/desempilhar do nosso dia a dia. Assim como a retirada de um elemento que não esteja no topo da Pilha.

O número de operações que pode ser realizado com uma Pilha é muito pequeno, tendo em vista o critério LIFO.

Três operações são consideradas básicas e, vou colocá-las no início, mas as demais também são necessárias para que as primeiras possam funcionar perfeitamente ou para fornecer alguma informação.

Inicializa() ou Init() - Inicializa a Pilha

Empilhar() ou Push() - Insere no topo da Pilha um elemento

Desempilhar() ou Pop() - Remove do topo da Pilha um elemento

acessoTopo() ou Top() - Tem acesso ao elemento que se encontra no topo da Pilha

verificaPilhCheia() ou isFull() - Verifica se Pilha está cheia

verificaPilhVazia() ou isEmpty() - Verifica se Pilha está vazia

Atenção

Muitos autores não destacam as três últimas operações porque, normalmente, elas estão embutidas nas demais ou, se não estiverem, um comando if, ou de atribuição, pode resolver o problema.

Nós vamos construir trechos para uma Pilha que poderá armazenar até 5 números inteiros.

Estarei apresentando, quando existir, o trecho que chama a função e a função.

2 Inicializar

A inicialização da Pilha irá se resumir a um comando de atribuição, preparando-a para receber os elementos que serão inseridos. Como na Pilha, o topo, isto é, a posição mais alta da Pilha é de maior importância, uma variável sempre deverá conter este valor. Sendo assim, para inicializar uma Pilha, precisaremos colocar um valor que não seja possível ser índice de vetor na linguagem C++. Gostaria de enfatizar que a inicialização não significa zerar todos os elementos da Pilha, mas é como se deixássemos engatilhado um ponteiro que seria acionado assim que o primeiro valor entrasse na pilha. Em programação, usamos uma variável e a inicialização foi feita na declaração. Observe o trecho.

```
//Inicialização
int ..., topo = -1, ...;
```

3 Empilhar (Push)

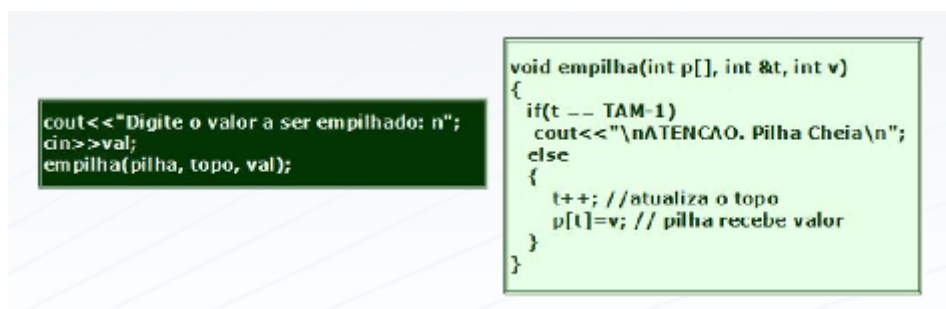
Esta é uma operação básica da Pilha. Para que possa ser empilhado um elemento, é preciso testar se existe lugar na Pilha.

Verificar se a Pilha está cheia poderia ser feito em separado, mas preferi fazer junto, pois se resume a um if, tornando-se mais rápido do que chamar uma função.

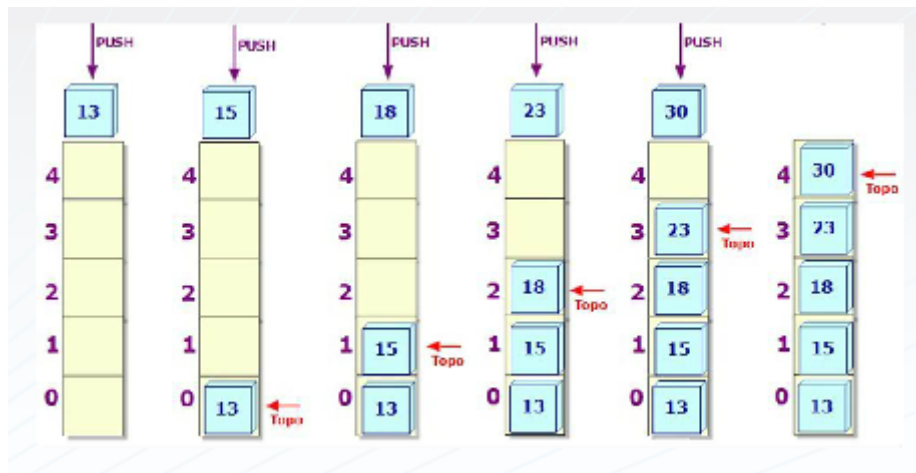
Além disso, uma chamada de função implica em colocar o endereço da instrução seguinte em uma Pilha para que o retorno seja possível e, se a função for algo muito simples, talvez não valha a pena ser criada. Foi assim que pensei e por essa razão reunimos duas operações nesta função. Observe.

Atenção

A função recebe o endereço do vetor, o endereço da variável que contém a posição do elemento que se encontra no topo e o valor a ser inserido. Os dois primeiros parâmetros precisam ter seus endereços passados porque serão alterados dentro da função. Observe que antes de inserir na Pilha, a variável *t* é incrementada porque o topo passa a ser o valor que será inserido na instrução abaixo. É uma função de fácil implementação.



Acompanhe agora, na figura abaixo, supondo que você digitou cinco números para serem empilhados.

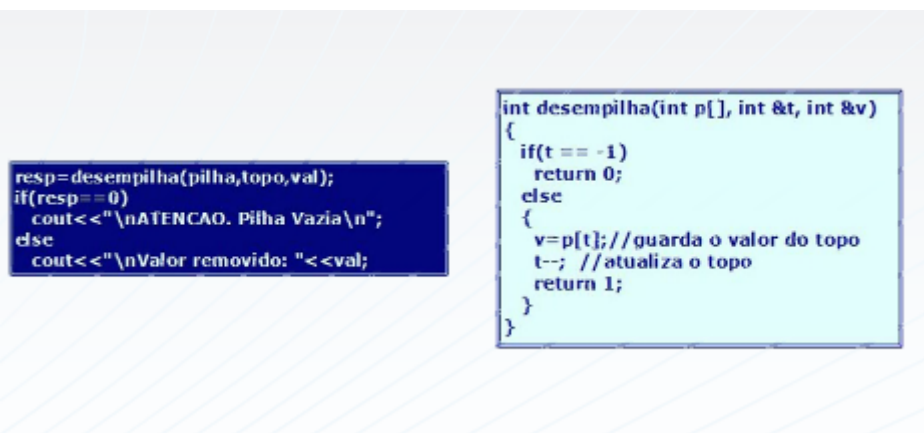


4 Desempilhar (Pop)

Esta é outra operação básica da Pilha. Para que possa ser desempilhado um elemento, é preciso testar se a Pilha não está vazia.

Verificar se a Pilha está vazia poderia ser feito em separado, mas preferi fazer junto, pois se resume a um if, tornando-se mais rápido do que chamar uma função.

Esta função é do tipo int, mas poderia ser do tipo void.



Atenção

A função recebe o endereço do vetor, o endereço da variável que contém a posição do elemento que se encontra no topo e o endereço da variável que vai receber o valor que será “desempilhado”.

Todos os parâmetros precisam ter seus endereços passados porque serão alterados dentro da função.

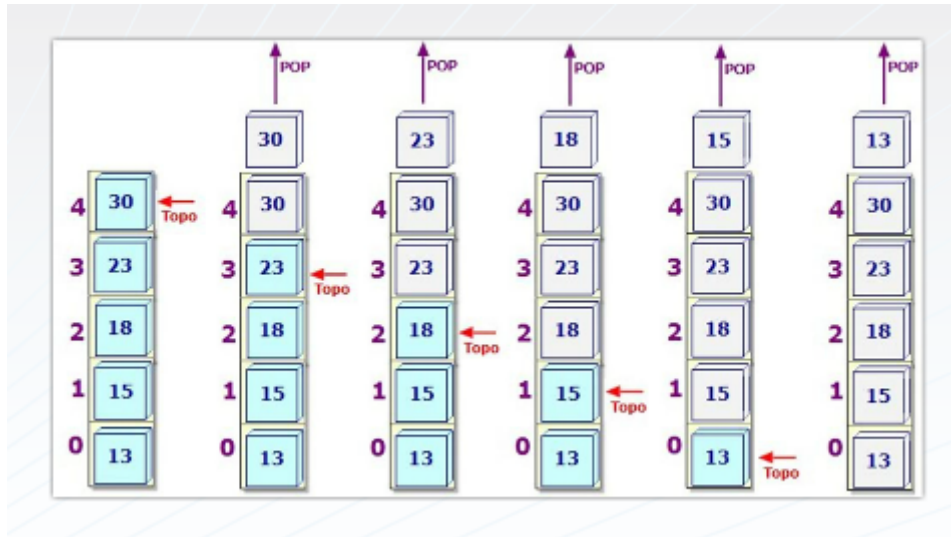
Observe que depois de copiar o conteúdo do elemento que estava no topo da *Pilha*, a variável *t* é decrementada.

Em nenhum momento foi removido o valor que estava armazenado na posição. Só o topo que se deslocou.

Sanou sua dúvida agora?

Assim como a função *empilhar*, a função *desempilhar* é de fácil implementação e bem parecida.

Acompanhe a agora, na figura abaixo, supondo que você escolheu desempilhar cinco números.



5 Acesso ao topo (Pop)

A única posição que poderá ser acessada na Pilha é a que está no topo da Pilha. Esta é uma função que possibilita que você visualize o elemento do topo sem “removê-lo”.

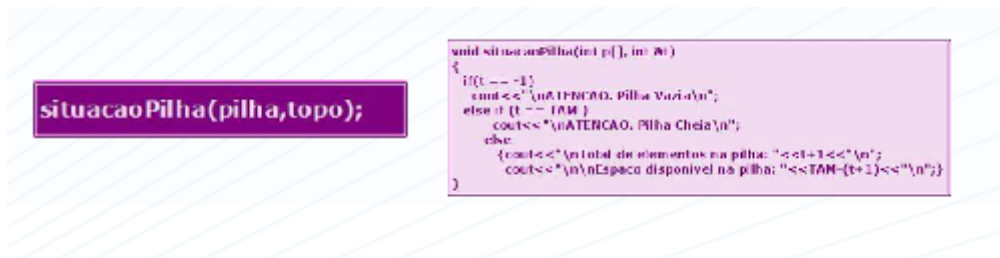
Na verdade, você não precisaria criar esta função tal sua simplicidade. Poderia simplesmente colocar no case. Só criei para modularizar todos os casos.

`acessoTopo(pilha,topo);`

```
void acessoTopo(int p[], int &t)
{
    if(t == -1)
        cout<<"\nATENCAO. Pilha Vazia\n";
    else
        cout<<"\nElemento do Topo da PILHA: "<<p[t];
}
```

Como todas as seis operações já foram apresentadas, vou apresentar mais uma. Nada importante, mas que poderá ser útil em algum momento.

Assim como a função anterior, você não precisaria criá-la.



Atenção

Esta função testa se a Pilha está cheia ou se a Pilha está vazia e, se não for nenhuma das duas opções, exibirá o total de elementos na Pilha e o espaço disponível.

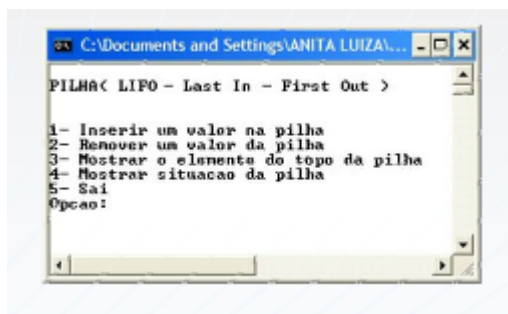
Você poderá escolher qual dos dois desejará exibir.

Clique aqui para baixar o exemplo em TXT.:

<http://estaciODOcente.webaula.com.br/cursos/gon119/docs/acessoao topo.txt>

Acredito que após termos visto todas as operações realizadas com a *Pilha*, poderemos compreender porque a Estrutura de Dados *Pilha* é considerada tão simples. Na verdade, suas operações são de fácil implementação.

Depois de termos visto as chamadas das funções e as funções, vamos ver o código fonte e acompanhar, rapidamente a execução do programa.



```

basica.cpp
1 #include <iostream>
2 #define TAM 5
3 using namespace std;
4 void empilha(int p[], int &t, int v);
5 int desempilha(int p[], int &t, int &v);
6 void acessoTopo(int p[], int &t);
7 void situacaoPilha(int p[], int &t);
8 int main()
9 {
10     int op, val, topo=-1, pilha[TAM], resp;
11
12     do
13     {
14         system("cls");
15         system("color f0");
16         cout<<"\nPILHA( LIFO - Last In - First Out )\n\n";
17         cout<<"\n1- Inserir um valor na pilha";
18         cout<<"\n2- Remover um valor da pilha";
19         cout<<"\n3- Mostrar o elemento do topo da pilha";
20         cout<<"\n4- Mostrar situacao da pilha";
21         cout<<"\n5- Sai";
22         cout<<"\nOpcao: ";
23         cin>>op;
24         system("cls");
25         switch(op)
26         {
27             case 1: cout<<"Digite o valor a ser empilhado: ";
28                     cin>>val;
29                     empilha(pilha, topo, val);
30                     break;
31
32             case 2: resp=desempilha(pilha,topo,val);
33                     if(resp==0)
34                         cout<<"\nATENCAO. Pilha Vazia\n";
35                     else
36                         cout<<"\nValor removido: "<<val;
37                     break;
38
39             case 3: acessoTopo(pilha,topo);
40                     break;
41
42             case 4: situacaoPilha(pilha,topo);
43                     break;
44
45             case 5: cout<<"\nPrograma basico da PILHA\n";
46                     break;
47
48             default: cout<<"\nOPCAO :INVALIDA\n";
49
50         }
51         cout<<"\n\n";system("pause!");
52     }while(op!=5);
53 }
54
55 /* Insere */
56 void empilha(int p[], int &t, int v)
57 {
58     if(t == TAM-1)
59         cout<<"\nATENCAO. Pilha Cheia\n";
60     else
61     {
62         t++; //atualiza o topo
63         p[t]=v; // pilha recebe valor
64     }
65 }
66
67 /* Remove */
68 int desempilha(int p[], int &t, int &v)
69 {
70     if(t == -1)

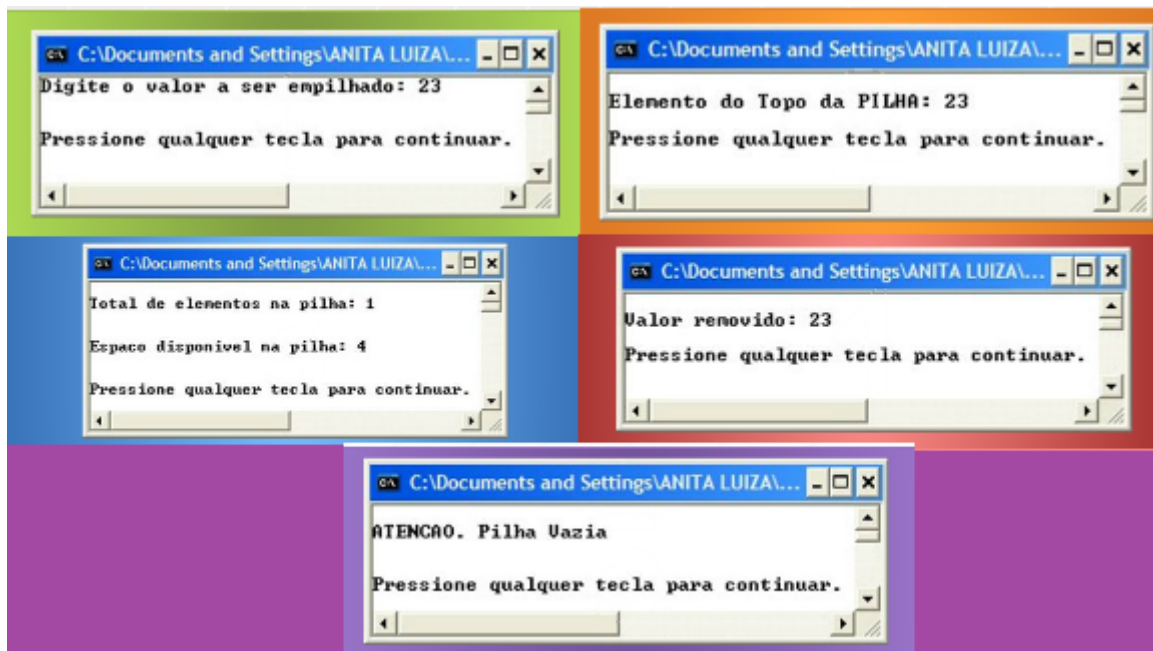
```



```

69     return 0;
70 else
71 {
72     v=p[t]; //guarda o valor do topo
73     t--; //atualiza o topo
74     return 1;
75 }
76 }
77
78 /* Mostra o topo */
79 void acessoTopo(int p[], int &t)
80 {
81     if(t == -1)
82         cout<<"\nATENCAO. Pilha Vazia\n";
83     else
84         cout<<"\nElemento do Topo da PILHA: "<<p[t];
85 }
86
87 /* Mostra situacao da Pilha */
88 void situacaoPilha(int p[], int &t)
89 {
90     if(t == -1)
91         cout<<"\nATENCAO. Pilha Vazia\n";
92     else if (t == TAM - 1)
93         cout<<"\nATENCAO. Pilha Cheia\n";
94     else
95         { cout<<"\nTotal de elementos na pilha: "<<t+1<<"\n";
96           cout<<"\n\nEspaco disponivel na pilha: "<<TAM-(t+1)<<"\n"; }
97 }

```



6 Aplicações com Pilhas

Depois de termos entendido o conceito de empilhar e desempilhar, acredito que seja mais fácil perceber a presença do uso desta estrutura de dados em várias aplicações.

- Histórico de páginas visitadas num navegador.
- Sequência de desfazer em vários softwares, o famoso atalho Ctrl Z.
- Implementação de recursividade (a torre de Hanói que vimos na disciplina de Algoritmos).
- A cadeia de chamadas de funções num programa.
- Avaliação de expressões aritméticas.
- Conversão de Decimal para Binário, etc.

Sobre o assunto de notações, já que não está incluído na ementa, falarei rapidamente sobre ele.

Na aritmética, estamos acostumados com a notação infixa onde o operador é colocado entre dois operandos.

Esta notação é problemática, pois requer conhecimento da hierarquia das operações, obrigando o uso dos parênteses para sobrepor esta hierarquia quando necessário.

Existem duas outras notações, a prefixa e a posfixa.

Na notação prefixa(notação polonesa), o operador antecede os operandos enquanto que na notação posfixa(notação polonesa reversa), os operandos antecedem o operador. A grande vantagem dessas notações é que elas não precisam de parênteses.

prefixa	infixa	posfixa
* 4 3	4 * 3	4 3 *

Como a calculadora será posfixa, nosso enfoque ficará com esta notação.

Exemplo 1

Infixa 4*3+2	Posfixa 4 3 * 2 +
------------------------	-----------------------------

Procure, a partir da esquerda para direita, colocar os operandos na ordem em que aparecem na expressão.

Quanto aos operadores, eles devem estar depois dos seus operandos.

Observe a expressão deste exemplo: tem duas operações onde a primeira tem hierarquia maior do que a segunda logo, a operação $4 * 3$ é executada primeiro:

Depois, o resultado será o primeiro operando da segunda operação.

Observe a tabela abaixo.

4 3 * 2+	Expressão posfixa
4 3 *	Primeira operação
12	Resultado da operação
12 2+	Segunda operação
14	Resultado

Atenção

Muita atenção. Algumas vezes, você poderá ter dois operadores juntos, pois tudo dependerá do número de operandos e da hierarquia das operações. Fique ligado.

Como a calculadora será posfixa, nosso enfoque ficará com esta notação.

Exemplo 2

Infixa 4*3 - 4/2	Posfixa 4 3 * 4 2 / -
-----------------------------------	--

Neste exemplo, temos três operações. Duas delas têm a mesma hierarquia. Acompanhe no quadro como você deverá colocar esta expressão na notação posfixa.

4 3 * 4 2 / -	Expressão posfixa
4 3 * 4 2 /	Primeira/Segunda operações
12 2	Resultados das operações
12 2 -	Terceira operação
10	Resultado

Vamos praticar!?

Atenção para os exemplos. Tente fazer e, depois, confira clicando no botão solução.

Infixa
18 - 4 * 3

Solução

Posfixa
18 4 3 * -

Infixa
6 * (5 - 3)

Solução

Posfixa
6 5 3 - *

Infixa
(8 - 2) / (2 + 1)

Solução

Posfixa

8 2 - 2 1 + /

7 Primeira Aplicação: Calculadora posfixa das quatro operações básicas


```

1 #include <iostream>
2 #include <cstdlib>
3 #define TAM 100
4 using namespace std;
5 void push(float p[], int &t, float v);
6 float pop(float p[], int &t);
7 int main()
8 {
9     int t = -1;
10    float a, b, p[TAM];
11    char s[10];
12
13    system("color f1");
14    cout<<"\n*****";
15    cout<<"\n*      Calculadora para quatro operacoes pos-fixa      *";
16    cout<<"\n*      Digite numeros ou operadores      *";
17    cout<<"\n*      Digite s para sair      *";
18    cout<<"\n*****\n";
19
20    do
21    {
22        cout<<" : ";
23        cin>>s;
24        switch(s[0])
25        {
26            case '+':
27                a = pop(p, t);
28                b = pop(p, t);
29                cout<<"\n"<< a+b<<"\n";
30                push(p, t, a+b);
31                break;
32
33            case '-':
34                a = pop(p, t);
35                b = pop(p, t);
36                cout<<"\n"<< b-a<<"\n";
37                push(p, t, b-a);
38                break;
39
40            case '*':
41                a = pop(p, t);
42                b = pop(p, t);
43                cout<<"\n"<< a*b<<"\n";
44                push(p, t, b*a);
45                break;
46
47            case '/':
48                a = pop(p, t);
49                b = pop(p, t);
50                if(a==0)
51                    cout<<"\ndivisao por 0\n";
52                else
53                {
54                    cout<<"\n"<< b/a<<"\n";
55                    push(p, t, b/a);

```



```

56         }
57         break;
58
59         default: push(p,t, atof(s));
60     }
61     } while(s[0]!='s');
62     system("pause");
63 }
64
65 /* Insere o elemento na pilha */
66 void push(float p[], int &t, float v)
67 {
68     if(t==TAM-1)
69         cout<<"\nATENCAO. Pilha Cheia\n";
70     else
71     {
72         t++; //atualiza o topo
73         p[t]=v; // pilha recebe valor
74     }
75 }
76
77 /* Remove o elemento da pilha */
78 float pop(float p[], int &t)
79 { float v;
80     if(t == -1)
81     {
82         cout<<"\nATENCAO. Pilha Vazia\n";
83         return 0;
84     }
85     else
86     {
87         v=p[t]; //guarda o valor do topo
88         t--; //atualiza o topo
89         return v;
90     }
91 }
92
93 /*Este programa é uma adaptação do código do livro C Completo e Total
94 de Herbert Schildt pagina 538.Como as funções eram com ponteiros, fiz
95 as alterações necessárias, mantendo as funções dos outros exemplos*/

```

```

C:\Documents and Settings\ANITA LUIZA\Meus documentos\calculadoraReal.exe
*****
*      Calculadora para quatro operacoes pos-fixa      *
*      Digite numeros ou operadores                    *
*      Digite s para sair                              *
*****
: 10
: 4
: 2.5
: 4
: *

10
: +

14
: *

140

```

Qual seria a expressão que representaria as operações realizadas?

Pense, escreva e depois confira passando o mouse no botão solução.

Solução

10 *(4+ 2.5*4)

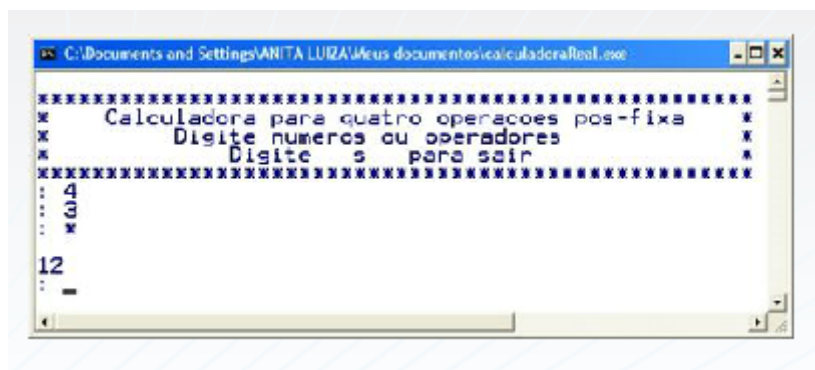
Considerações sobre o programa

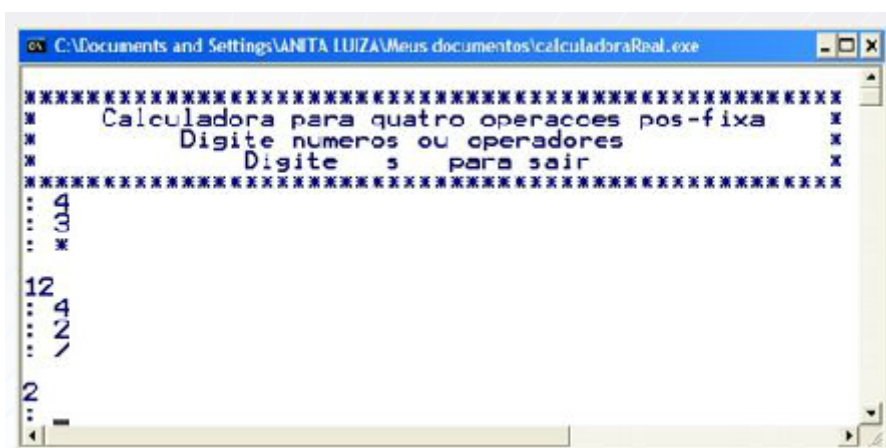
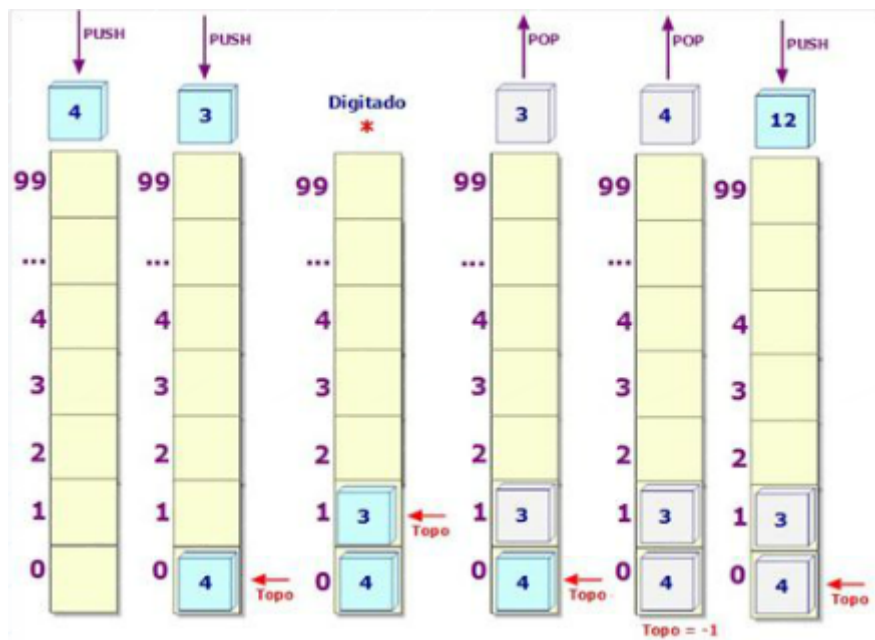
As operações push e pop são as mesmas do programa da pilhaBasica.cpp exceto pelos nomes.

Nesta aplicação, cada vez que um operador é encontrado, dois elementos da *Pilha* são desempilhados e operados e o resultado da operação retorna para a *Pilha*.

Vamos acompanhar o passo a passo da notação reversa polonesa com a expressão abaixo:

Posfixa
4 3 * 4 2 / -





8 Segunda Aplicação: Conversão de um número decimal para binário

```

1 #include <iostream>
2 #define TAM 40
3 using namespace std;
4 void empilha(int p[], int &t, int v);
5 int desempilha(int p[], int &t, int &v);
6 int main()
7 { float n; // para possibilitar a entrada de número maior do que o inteiro permite
8   int num, resto, pilha[TAM], sinal, topo=-1; //Inicialização da pilha através de topo=-1
9   system("color 2f");
10  cout<<"\n#####";
11  cout<<"\n###";
12  cout<<"\n### Converta Numero da base decimal para base binaria ###";
13  cout<<"\n###";
14  cout<<"\n#####\n";
15  cout<<"\nDigite numero positivo ate 2147483520. Qualquer outro, sai: ";
16  cin>>n;
17  if(n > 2147483520 || n < -2147483520)
18    exit(0); //Limita intervalo de inteiro no Dev-Cpp, embora seja maior
19  else
20  {
21    num=(int)n; //converte real para inteiro
22    while(num>=0)
23    {
24      do //inicio do trecho que empilha os restos que irão gerar o número binário
25      {
26        resto= num%2;
27        empilha(pilha, topo, resto);
28        num/=2;
29      }while(num > 0); //fim do trecho de empilhamento
30      cout<<"\nConvertido para binario: ";
31      sinal=desempilha(pilha, topo, resto); //inicio do trecho que desempilha todos
32      while(sinal == 1) //os restos que irão exibir o número binário
33      {
34        cout<<resto;
35        sinal=desempilha(pilha, topo, resto);
36      } //Fim do trecho de desempilhamento
37      topo=-1;
38      cout<<"\n\nDigite numero positivo ate 2147483520. Qualquer outro, sai: ";
39      cin>>n; if(n > 2147483520 || n < -2147483520) exit(0); //Máximo permitido no Dev Cpp
40      else num=(int)n;
41    }
42    cout<<"\n\n";
43    system("pause");
44  }
45 void empilha(int p[], int &t, int v)
46 {
47   if(t==TAM-1) //Este teste não é necessário porque já limitei a entrada
48     cout<<"\nATENCAO. Pilha Cheia\n"; // Mantive para usar o mesmo trecho
49   else
50   {
51     t++; //atualiza o topo
52     p[t]=v; // pilha recebe valor
53   }
54 }
55
56 int desempilha(int p[], int &t, int &v)
57 {
58   if(t == -1)
59     return 0;
60   else
61   {
62     v=p[t]; //guarda o valor do topo
63     t--; //atualiza o topo
64     return 1;
65   }
66 }

```


Considerações sobre o programa

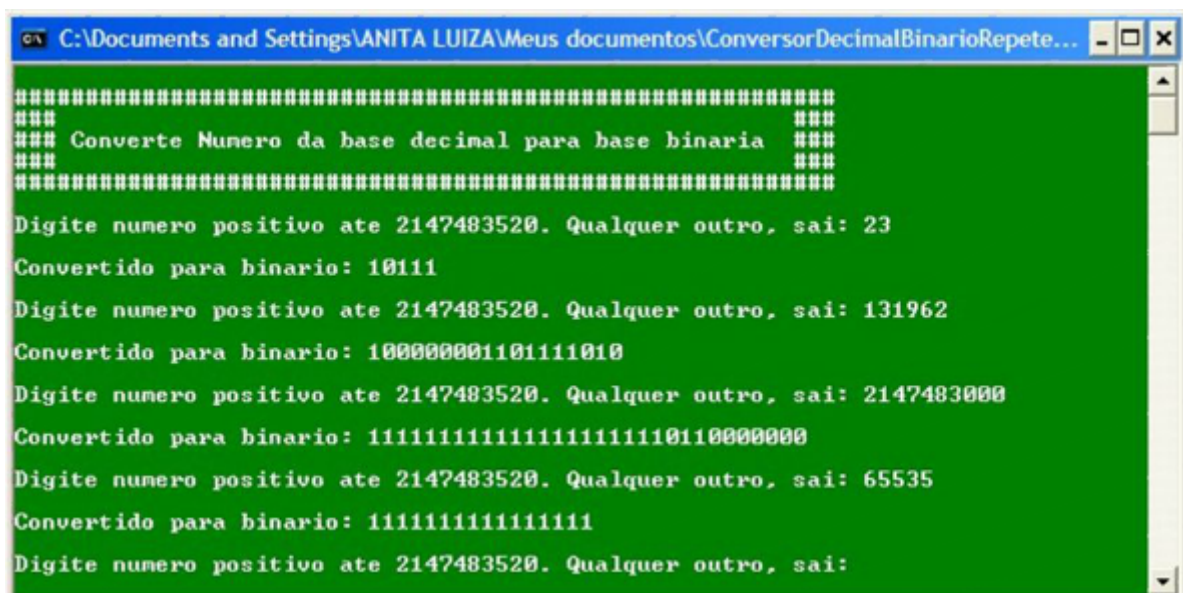
O tipo o inteiro está limitado 2 147 483 647 entretanto no Dev- Cpp, nesta versão, ele não consegue chegar a este valor.

Por essa razão, a escolha foi por uma variável do tipo real porque suporta valores maiores e por abandonar o programa quando o número estiver fora do intervalo dos números inteiros através da função exit(0).

Caso isso não aconteça, foi feita a conversão para inteiro e armazenado o valor na variável num.

O programa está todo comentado e os trechos são muito simples. Entretanto sugiro que, se continuar com dificuldade em compreender algum deles, releia a matéria de Algoritmos e se a dúvida persistir, fale com seu professor.

Hoje nós ficamos por aqui, mas será que você vai preferir ser o primeiro a chegar e o primeiro a sair? Só na próxima aula que você poderá decidir. No momento, você fica com o critério de quem chega por último, sai primeiro.



```
#####  
### Converte Numero da base decimal para base binaria ###  
#####  
Digite numero positivo ate 2147483520. Qualquer outro, sai: 23  
Convertido para binario: 10111  
Digite numero positivo ate 2147483520. Qualquer outro, sai: 131962  
Convertido para binario: 100000001101111010  
Digite numero positivo ate 2147483520. Qualquer outro, sai: 2147483000  
Convertido para binario: 111111111111111110110000000  
Digite numero positivo ate 2147483520. Qualquer outro, sai: 65535  
Convertido para binario: 11111111111111  
Digite numero positivo ate 2147483520. Qualquer outro, sai:
```

Espero que você tenha compreendido o conceito da Estrutura de Dados Pilha e que as duas aplicações mostradas na aula lhe motive para criar programas usando esta estrutura e enviando para o grupo.

Não é uma aula com muito conteúdo porque a aula anterior já deu embasamento para esta e outras aulas.

Fora o conceito simples da Pilha, nada foi novidade porque já estudamos tudo que usamos.

Peço, como em todas aulas, que não deixe pendente nenhum assunto. Tente fazer os exercícios da lista e procure seu professor toda vez que tiver alguma dúvida.

Até a próxima aula!

Saiba mais



Clique no link a seguir e baixe um arquivo da aula:

http://estaciocodocente.webaula.com.br/cursos/gon119/docs/ED06_DOC.pdf

O que vem na próxima aula

Na próxima aula, você estudará os seguintes assuntos:

- Conceituar a estrutura de dados Fila;
- Representar a estrutura de dados Fila por contiguidade (Fila simples);
- Compreender e implementar as operações com Fila simples;
- Analisar aplicações com Filas simples.

CONCLUSÃO

Nesta aula, você:

- Compreendeu a importância da Estrutura de Dados *Pilha*;
- Compreendeu e usou várias operações que podem ser feitas com a *Pilha*;
- Analisou aplicações da Pilha e acompanhou a execução de programas.