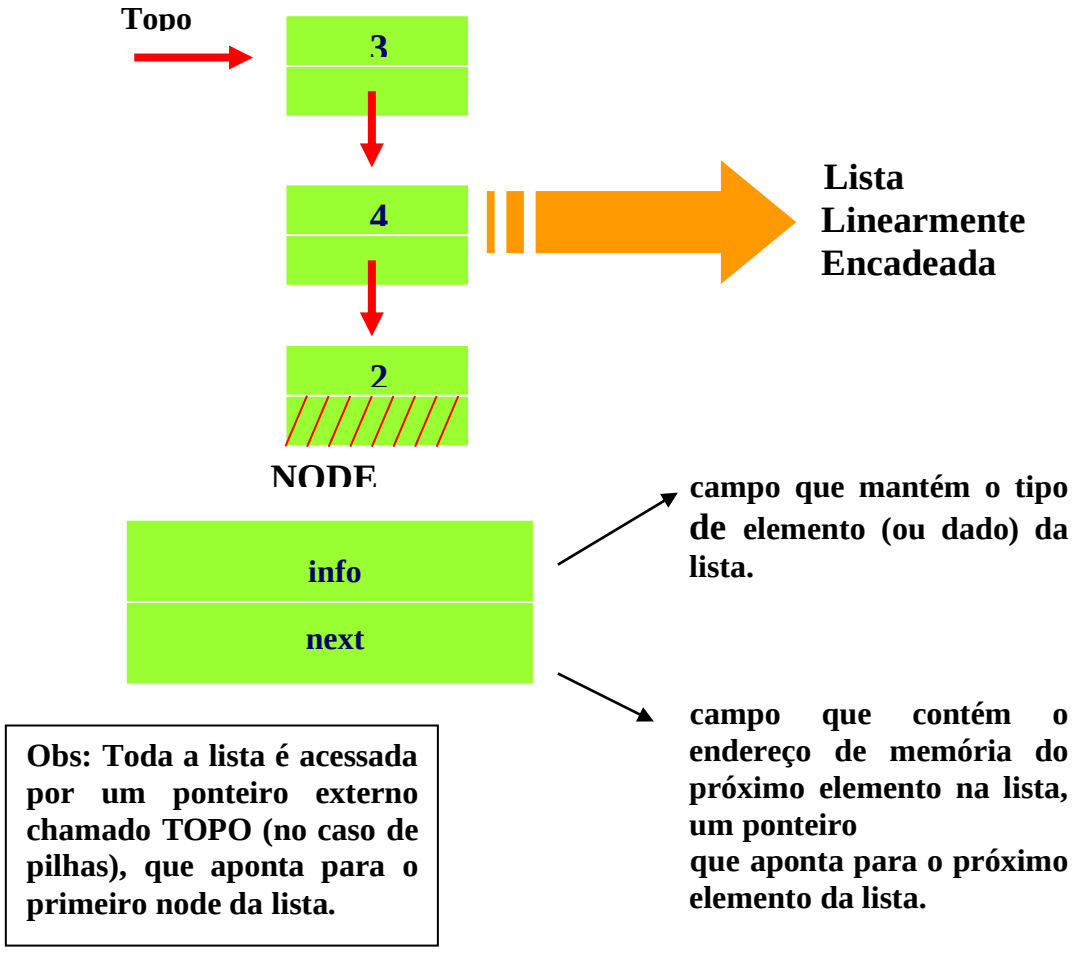
Listas Lineares Ligadas

A memória ainda é um recurso escasso e de preço elevado nos sistemas de computação. Portanto, existe a preocupação de encontrar um mecanismo que faça a alocação de memória somente no momento realmente necessário para armazenar alguma informação, e que exista também um processo de liberação dessa área de memória quando não se precisa mais dessa informação, tornando possível seu uso para a realização do armazenamento de outras informações. Esse mecanismo pode estar disponível por meio do uso das listas ligadas.

Com o uso de listas ligadas não é necessário usar áreas de memória contígua, as listas ligadas utilizam a chamada alocação encadeada, que pressupõe o uso de nós. Cada nó contém duas partes distintas, chamadas de campos; o primeiro campo contém a informação propriamente dita, e o segundo campo contém o endereço do próximo nó.

Se soubermos o endereço do primeiro nó, o segundo é atingido utilizando o endereço que estará no primeiro nó, e assim sucessivamente até chegarmos no último nó, quando o endereço será nulo. O endereço do primeiro nó é armazenado em uma variável externa sob forma de um ponteiro.

Uma lista ligada sem nós é uma lista vazia ou nula. Neste caso, o ponteiro externo que aponta para o início da lista ligada é de valor nulo.



Alocação Dinâmica de Memória

É o meio pelo qual um programa pode obter memória durante seu tempo de execução. A área de alocação dinâmica consiste em toda memória disponível que não foi usada para outro propósito. Esta área é comumente referida como **heap**. Em outras palavras, o **heap** é o resto da memória.
Em C++, alocação dinâmica de memória é feita através dos comandos **new** e **delete**. O operador **new** retorna um ponteiro para a área da memória alocada. O operador **delete** libera a memória previamente alocada por **new**.

Exemplo de Alocação de Memória

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main( )
{

   int *p = new int;                          // "aloque" espaço para um inteiro
   if(p == NULL) {                            // não há espaço na memória
   cout << "Erro de alocação\n"; exit(1);
    }
    *p = 100;
    cout << "Na posição " << p << "o valor é : " << *p << endl;
    delete p;                                 // liberando o espaço alocado

    system("PAUSE");
    return 0;
}
```

O Ponteiro *this*

Considere uma classe em C++ definida com seus atributos (*data members*) e operações (*member functions*). Considere ainda que um objeto da classe foi criado e que o usuário solicita ao objeto para executar uma de suas operações. Neste momento, o computador aloca, automaticamente, um ponteiro para este objeto. Este ponteiro é conhecido como o ponteiro **this**. Em outras palavras, o ponteiro **this** aponta sempre para o objeto que está sendo solicitado para executar uma de suas operações, durante o tempo de execução de um programa.

Desvantagens de Arrays para a implementação de Pilhas:

O uso de arrays para a simulação (ou implementação) de pilhas apresenta as seguintes desvantagens:

- ✓ Uma quantidade fixa de memória é alocada sem, necessariamente, ser usada.
- ✓ Uma vez definido o tamanho do array, nenhuma quantidade adicional de memória pode, teoricamente, ser alocada para aquele array.

Classes em C++

A base de um programa em C++ são as classes (Class). Elas funcionam de forma muito parecida com as estruturas (struct), diferenciando-se por criar elementos públicos (**public**) ou privados (**private)**.

Em uma estrutura todos os elementos são públicos, podem ser acessados por qualquer parte do programa (desde que esteja dentro do escopo da estrutura), já os membros privados da classe podem apenas ser acessados de dentro da própria classe.

Exemplo:

```
Class Classe1
{
private:
        int x,y;
public:
        void exibe_valores( ) {
                cout x << y ;
        }
};
```

Obs: Somente a função exibe_valores( ) pode ser acessada pelo resto do programa, os valores x e y são privados para o uso da classe.

Implementação de Pilhas utilizando Lista Ligada

```
#include <cstdlib>
#include <iostream>

using namespace std;

class Pilha {
  int     info;    //membros da classe pilha
  Pilha   *next;  //ponteiro que armazena o próximo endereço
public:
  Pilha();                  //construtor: sempre é executado quando criamos um objeto
  void    Push(int n);        //inserir elemento
  int     Pop();             //remover elemento
  void    Imprime();          //listar conteúdo da pilha
};
Pilha        *TOPO;         //Tipo do ponteiro externo: Pilha, que representa o nodo
```

```
Pilha::Pilha()
{
  info = 0;
  next = NULL;
}

void Pilha::Push(int n)
{
  this -> info = n;
  this -> next = TOPO;
  TOPO = this;
}

int   Pilha::Pop()
{
   int    temp;
   if(TOPO == NULL)
     {
       cout << "Pilha Vazia" << endl;
       return(0);
     }
   temp = this -> info;
   TOPO = this -> next;
   delete(this);  //destrutor da classe
   return(temp);
}

void  Pilha::Imprime()
{
   if(TOPO == NULL)
     cout << "Pilha Vazia" << endl;
   else {
      Pilha    *temp;
      temp = TOPO;
      while(temp != NULL)
     {
         cout << temp -> info << endl;
         temp = temp -> next;
     }
   }
}
```

```
int main( )
{
{
  system("CLS");

 Pilha    *pp;
 int    valor;
 int    escolha;

 TOPO = NULL;

 do {
   system("CLS");

    cout<<"                MENU PRINCIPAL  \n\n\n";
    cout<<"\n             (1) - Insere um elemento na Pilha";
    cout<<"\n             (2) - Remove um elemento da Pilha";
    cout<<"\n             (3) - Imprime a Pilha";
    cout<<"\n             (4) - Para SAIR";
    cin>>escolha;
    switch (escolha)
{
    case  1:
           system("CLS");
          pp = new Pilha();
          cout << "\Entre com um numero : ";
          cin  >> valor;
          pp->Push(valor);
          break;
   case 2:
           system("CLS");
          valor = TOPO -> Pop();
          cout << "valor removido: " << valor << endl ;
           system("PAUSE");
          break;
  case 3:
         system("CLS");
         TOPO -> Imprime();
         system("PAUSE");
         break;
 case 4:
        exit(1);
        break;
 default :
        system("CLS");
       cout << "Leia as intrucoes";
       system("PAUSE");
```

```
 }

} while (escolha!=4);
}

   system("PAUSE");
   return 0;
}
```

Operações com listas ligadas

Inserção de um nó

1. Conseguir um endereço pp para um novo nó;
2. Em seguida, é necessário colocar a informação desejada no respectivo campo do nó apontado por pp.
3. A seguir, o endereço do primeiro nó é colocado no campo de endereço do nó recém-criado;
4. Finalmente, o conteúdo da variável externa TOPO é atualizado com o valor do ponteiro presente em pp.

Remoção de um nó

1. Primeiramente, o valor do endereço armazenado em TOPO é guardado na variável temp;
2. A seguir, o conteúdo do campo endereço do nó a ser excluído é armazenado em TOPO;
3. Em seguida, o endereço dos nós a serem excluídos é fornecido como área de heap (memória livre) no sistema para uso futuro.