Introdução a ponteiros

Claudia Akemi Izeki

O que é um ponteiro?

- Variável cujo valor é um endereço de memória.

Operador de endereço (&)

- Obtém o endereço de uma variável
  - Escreva o operador de endereço & antes do nome da variável

Ex:

```
int x;

// Endereço da variável x na memória
cout << "&x: " << &x;
```

Operador de endereço (&)

- Obtém o endereço de uma variável
  - Escreva o operador de endereço & antes do nome da variável

Ex:

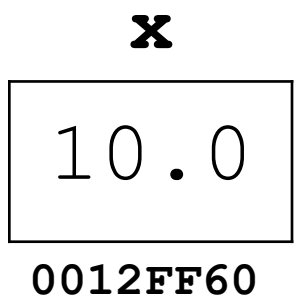
```
int x;
```

&x: 0012FF60

```
// Endereço da variável x na memória
cout << "&x: " << &x;
```

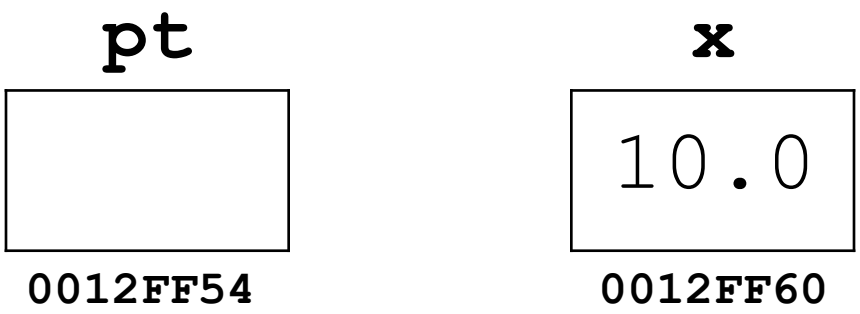
Exemplo

```
float x = 10.0;
```



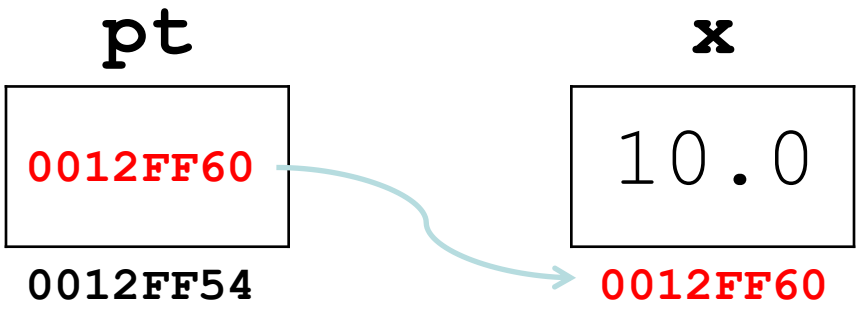
Exemplo

```
float x = 10.0;
float *pt;  // Declaração de um ponteiro com nome pt
            // Lê-se: "pt é um ponteiro para uma variável do
            //         tipo float"
```



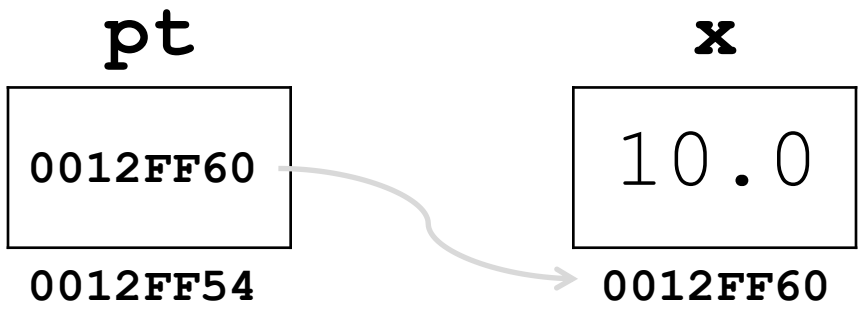
Exemplo

```
float x = 10.0;
float *pt;

pt = &x;  // pt recebe o endereço de x, ou seja,
          // aponta para a variável x, que é do tipo float
```



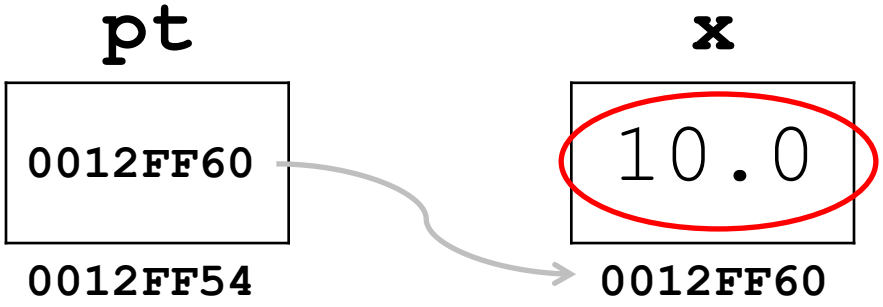
Exemplo

```
float x = 10.0;
float *pt;

pt = &x;

// Qual as impressões?
cout << "x: " << x;
cout << "pt: " << pt;
cout << "&x: " << &x;
cout << "&pt: " << &pt;
cout << "*pt: " << *pt;
```



Exemplo

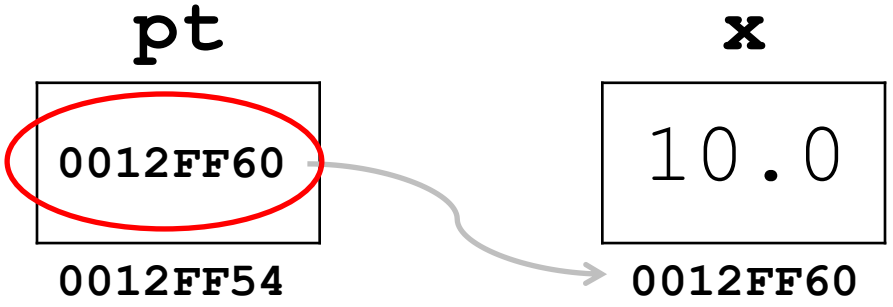
```
float x = 10.0;
float *pt;

pt = &x;

// Qual as impressões?
cout << "x: " << x;
cout << "pt: " << pt;
cout << "&x: " << &x;
cout << "&pt: " << &pt;
cout << "*pt: " << *pt;
```



Qual o valor (conteúdo) da variável x?

Exemplo

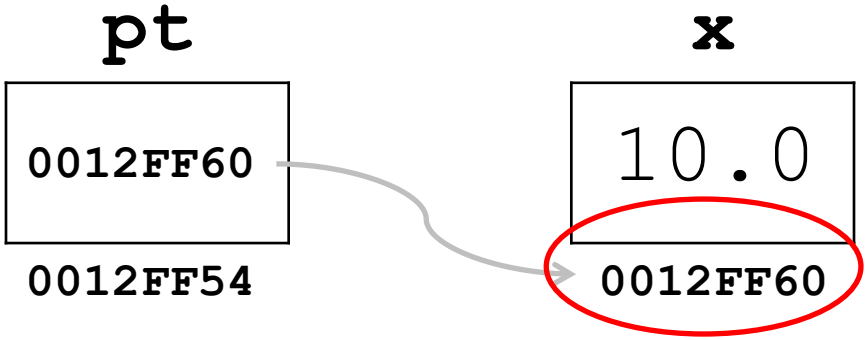
```
float x = 10.0;
float *pt;

pt = &x;

// Qual as impressões?
cout << "x: " << x;
cout << "pt: " << pt;
cout << "&x: " << &x;
cout << "&pt: " << &pt;
cout << "*pt: " << *pt;
```



Qual o valor (conteúdo) da variável `pt`?

Exemplo

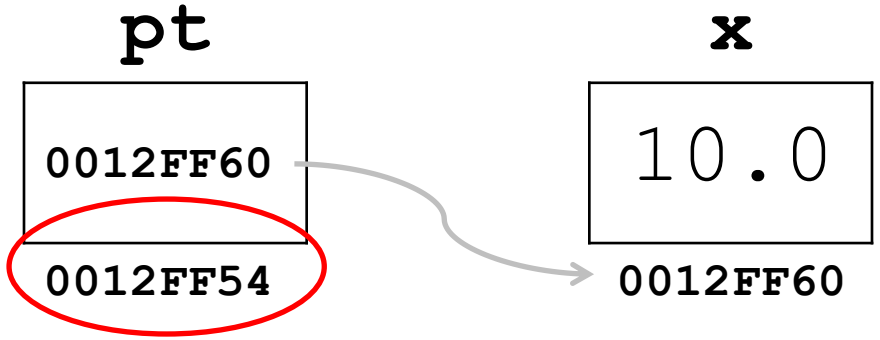
```
float x = 10.0;
float *pt;

pt = &x;

// Qual as impressões?
cout << "x: " << x;
cout << "pt: " << pt;
cout << "&x: " << &x;
cout << "&pt: " << &pt;
cout << "*pt: " << *pt;
```



Qual o endereço na memória da variável x?

Exemplo

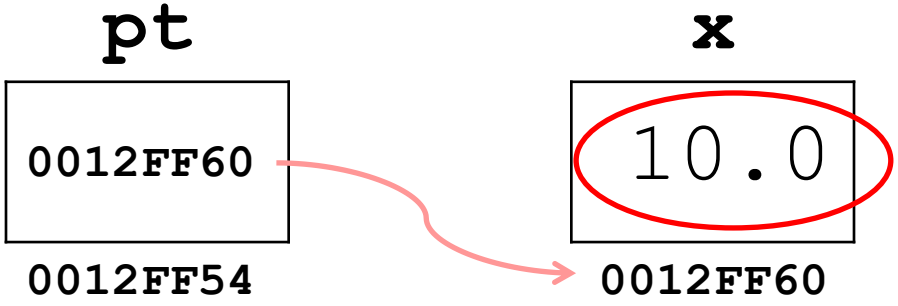
```
float x = 10.0;
float *pt;

pt = &x;

// Qual as impressões?
cout << "x: " << x;
cout << "pt: " << pt;
cout << "&x: " << &x;
cout << "&pt: " << &pt;
cout << "*pt: " << *pt;
```



Qual o endereço na memória da variável `pt`?

Exemplo

```
float x = 10.0;
float *pt;

pt = &x;

// Qual as impressões?
cout << "x: " << x;
cout << "pt: " << pt;
cout << "&x: " << &x;
cout << "&pt: " << &pt;
cout << "*pt: " << *pt;
```

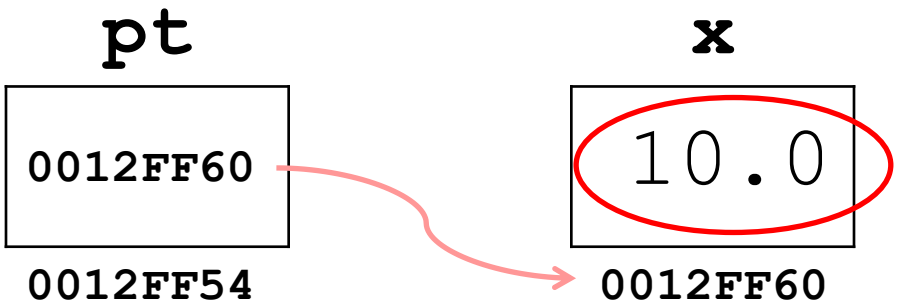


Qual o valor da variável apontada por `pt`?

Operador indireto (*)

- O operador indireto (*) antes do nome do ponteiro resulta no valor da variável apontada.
- Ex:

```
cout << "*pt: " << *pt;
```



Para que vamos usar ponteiros nesta disciplina?

1. Como alternativa à passagem de parâmetros por referência usando apelido;
2. Para a alocação dinâmica de estruturas como os vetores;
3. Principalmente para a manipulação (criação/alteração/acesso) de estruturas de dados alocadas dinamicamente, como listas encadeadas e árvores binárias.

Passando argumentos por referência usando apelido

- Exemplo:

```
void troca(int &x, int &y)
{
   int aux = x;
   x = y;
   y = aux;
}
```

```
int main()
{
   int a, b;

   cout << "Entre com os valores de a e b: ";
   cin >> a >> b;

   troca(a, b);

   cout << "Os valores de a e b foram trocados.";
   cout << "a: " << a;
   cout << "b: " << b;
    return 0;
}
```

Passando argumentos por referência usando ponteiros

- Exemplo:

```
int main()
{
   int a, b;

   cout << "Entre com os valores de a e b: ";
   cin >> a >> b;

   troca(___?___, ___?___);

   cout << "Os valores de a e b foram trocados.";
   cout << "a: " << a;
   cout << "b: " << b;
    return 0;
}
```

```
void troca(___?___, ___?___)
{
   ____?____ = ____?____;
   ____?____ = ____?____;
   ____?____ = ____?____;
}
```

Passando argumentos por referência usando ponteiros

- Exemplo:

```
void troca(________, ________)
{
   _________ = _________;
   _________ = _________;
   _________ = _________;
}
```

```
int main()
{
   int a, b;

   cout << "Entre com os valores de a e b: ";
   cin >> a >> b;

   troca(&a, &b);

   cout << "Os valores de a e b foram trocados.";
   cout << "a: " << a;
   cout << "b: " << b;
    return 0;
}
```

São passados os endereços das variáveis a e b

Passando argumentos por referência usando ponteiros

- Exemplo:

```
int main()
{
   int a, b;

   cout << "Entre com os valores de a
   cin >> a >> b;

   troca(&a, &b);

   cout << "Os valores de a e b foram trocados: ";
   cout << "a: " << a;
   cout << "b: " << b;
    return 0;
}
```

```
void troca(int *x, int *y)
{
   _________ = _____;
   _________ = _____;
   _________ =
}
```

x e y são ponteiros, ou seja, são variáveis usadas para armazenar os endereços enviados pela função chamadora.

Passando argumentos por referência usando ponteiros

- Exemplo:

```
int main()
{
   int a, b;

   cout << "Entre com os valores de a
   cin >> a >> b;

   troca(&a, &b);

   cout << "Os valores de a e b foram trocados.";
   cout << "a: " << a;
   cout << "b: " << b;
    return 0;
}
```

```
void troca(int *x, int *y)
{
   int aux = *x;
   _____ = _____;
   _____ ;
}
```

aux é uma variável do tipo int que recebe o conteúdo da variável apontada por x, que é do tipo int.

Passando argumentos por referência usando ponteiros

- Exemplo:

```
int main()
{
   int a, b;

   cout << "Entre com os valores de a
   cin >> a >> b;

   troca(&a, &b);

   cout << "Os valores de a e b foram
   cout << "a: " << a;
   cout << "b: " << b;
    return 0;
}
```

```
void troca(int *x, int *y)
{
   int aux = *x;
   *x = *y;
   _________ = _________;
}
```

O conteúdo da variável apontada por `x` é substituído pelo conteúdo da variável apontada por `y`.

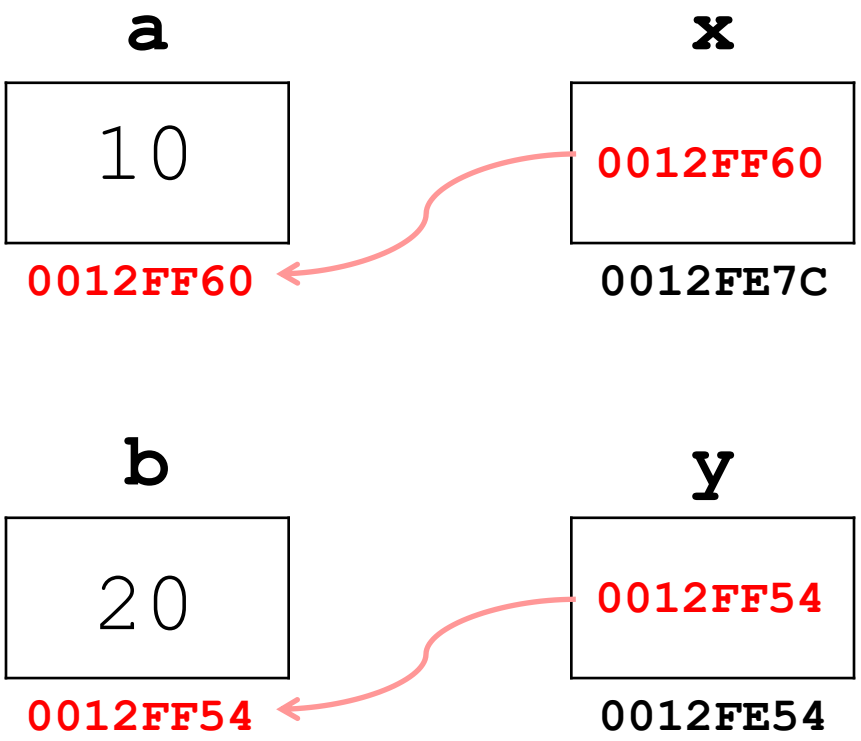
Passando argumentos por referência usando ponteiros

- Exemplo:

```
int main()
{
   int a, b;

   cout << "Entre com os valores de
   cin >> a >> b;

   troca(&a, &b);

   cout << "Os valores de a e b fora
   cout << "a: " << a;
   cout << "b: " << b;
    return 0;
}
```

```
void troca(int *x, int *y)
{
   int aux = *x;
   *x = *y;
   *y = aux;
}
```

O conteúdo da variável apontada por `y` é substituído pelo conteúdo da variável `aux`.

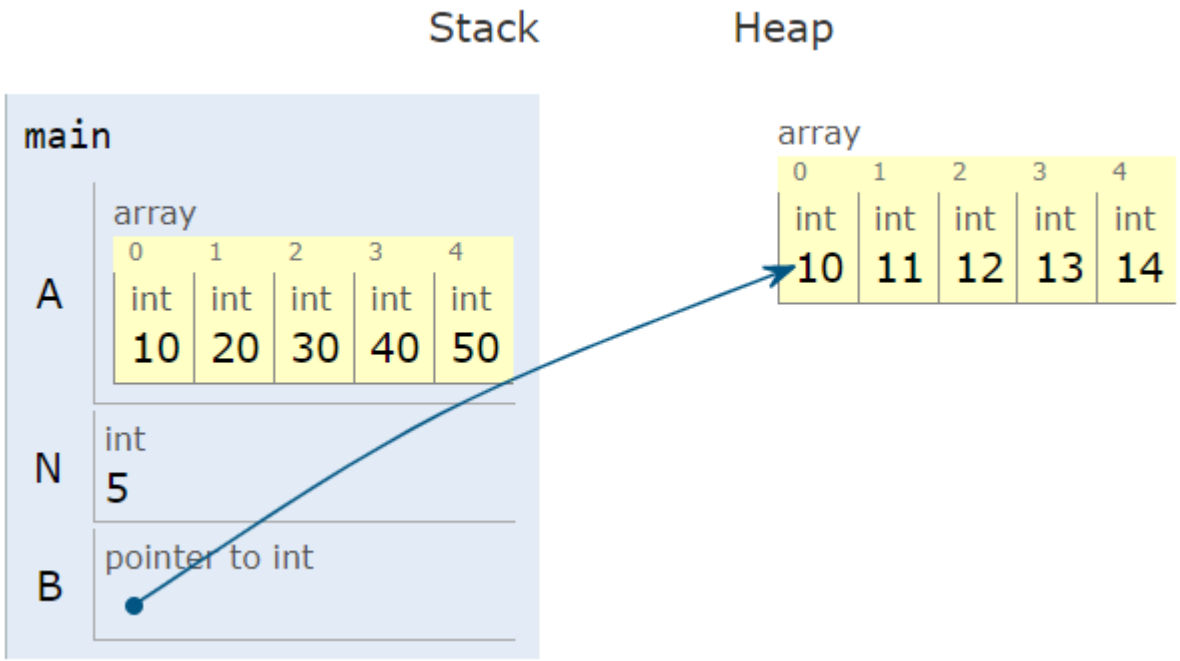

a
x
10
0012FF60
0012FF60
0012FE7C
b
y
20
0012FF54
0012FF54
0012FE54

Para que vamos usar ponteiros nesta disciplina?

1. Como alternativa à passagem de parâmetros por referência usando apelido;
2. Para a alocação dinâmica de estruturas como os vetores;
3. Principalmente para a manipulação (criação/alteração/acesso) de estruturas de dados alocadas dinamicamente, como listas encadeadas e árvores binárias.

Diferença entre alocação estática e dinâmica de um vetor

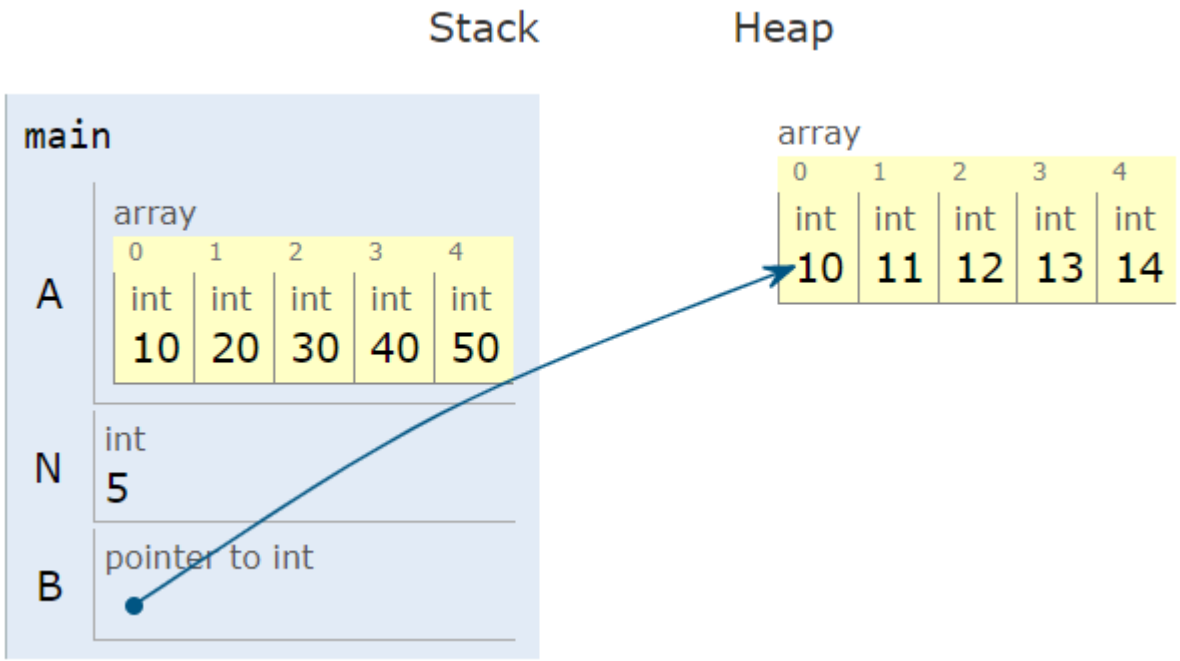
```
int A[5];
int N;
cin >> N;
int *B = new int [N];
```



Diferença entre alocação estática e dinâmica de um vetor

```
int A[5];
int N;
cin >> N;
int *B = new int [N];
```

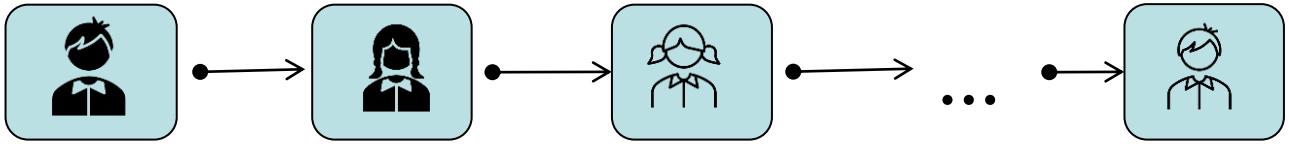
Na alocação estática, o espaço de memória das variáveis é definido durante a compilação na **stack**. Já na alocação dinâmica, é definido durante a execução do programa na **heap**.

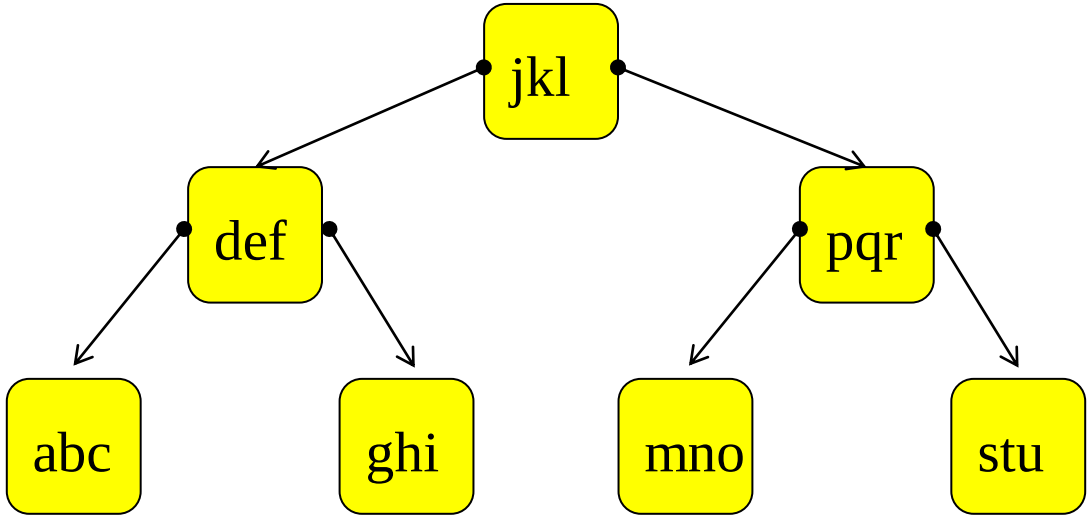


Para que vamos usar ponteiros nesta disciplina?

1. Como alternativa à passagem de parâmetros por referência usando apelido;
2. Para a alocação dinâmica de estruturas como os vetores;
3. Principalmente para a manipulação (criação/alteração/acesso) de estruturas de dados alocadas dinamicamente, como listas encadeadas e árvores binárias.

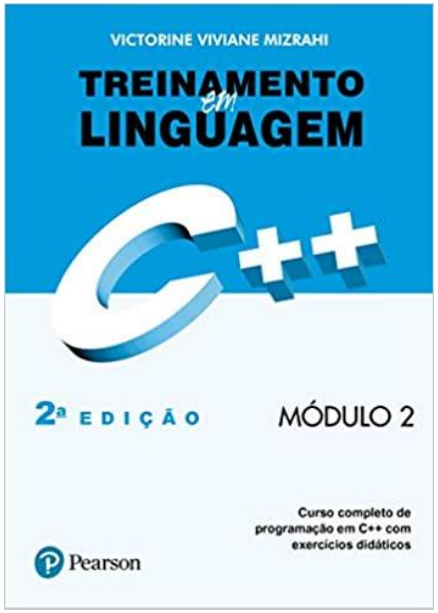
Estruturas de dados alocadas dinamicamente: listas encadeadas, árvores





Bibliografia

- Início do Capítulo 11 – Ponteiros (páginas 121 a 129) do livro:
  - MIZRAHI, V. V.Treinamento em linguagem C++: módulo 2. 2. ed. São Paulo: Makron books, 2006.



https://unifei.edu.br/ensino/bibliotecas

BIBLIOTECA VIRTUAL 3.0 UNIVERSITÁRIA

Bibliografia

- Início do Capítulo 8 – Ponteiros e *strings* baseadas em ponteiros (páginas 312 a 319) do livro:
  - DEITEL, H. M.; DEITEL, P. J. C++ Como Programar. 5. ed. São Paulo: Pearson Prentice Hall, 2006.

https://unifei.edu.br/ensino/bibliotecas

BIBLIOTECA VIRTUAL 3.0 UNIVERSITÁRIA