Conceitos c++

Prof. Demétrios Coutinho

INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
RIO GRANDE DO NORTE

Campus Pau dos Ferros
Disciplina de Organização de Algoritmos
Demetrios.coutinho@ifrn.edu.br

Por que implementar em C++?

- Liguagem Tradicional que deveria ser dominada por todo profissional de computação.

- Pode ser bastante eficiente.

- Aplicações que requer desempenho como: aplicações numéricas, computação gráfica software embarcado.

- SO feitos em C : Microsoft Windows, Mac OS X, GNU/Linux

- Liguagem de alto nivel feitos em C : Perl, PHP, Python, Matlab e outras.

C/C++

C é uma linguagem de programação criada por Dennis Ritchie, entre 1969 e 1973 no AT&T Bell Labs

C++, por outro lado, foi desenvolvida por Bjarne Stroustrup a partir de 1979, também no Bell Labs, e adiciona características de orientação a objetos, como classes, e outras melhorias à linguagem C.

Conceito de Memória

A memória de um computador é o local em que são guardadas informações necessárias para execução de programas;

Esta memória é volátil, isso é uma vez desligado o computador as informações contidas nela são perdidas;

Os algoritmos utilizam esta memória para salvar informações durante a sua execução;

Os locais na memória em que os algoritmos salvam suas informações são chamadas de variáveis.

VARIÁVEIS

Variáveis são utilizadas no algoritmo para se armazenar algum valor;

Podemos ver uma variável como uma gaveta onde guardamos nossas coisas, porém com algumas restrições:

- Cada gaveta só pode armazenar um “tipo de coisa”. Ex.: uma gaveta de camisas, outra de papéis, etc;
- Cada gaveta deve ter um nome que a identifique.

Cada gaveta faz parte de um conjunto de outras gavetas, as quais compõem a “memória”.

O valor (dado/informação) de uma variável é armazenada na memória.

- Permanece até que a “execução” do algoritmo termine.

VARIÁVEIS

Memória – Conjunto de locais para armazenar dados.

Dados podem ser armazenados na memória, por meio das variáveis.

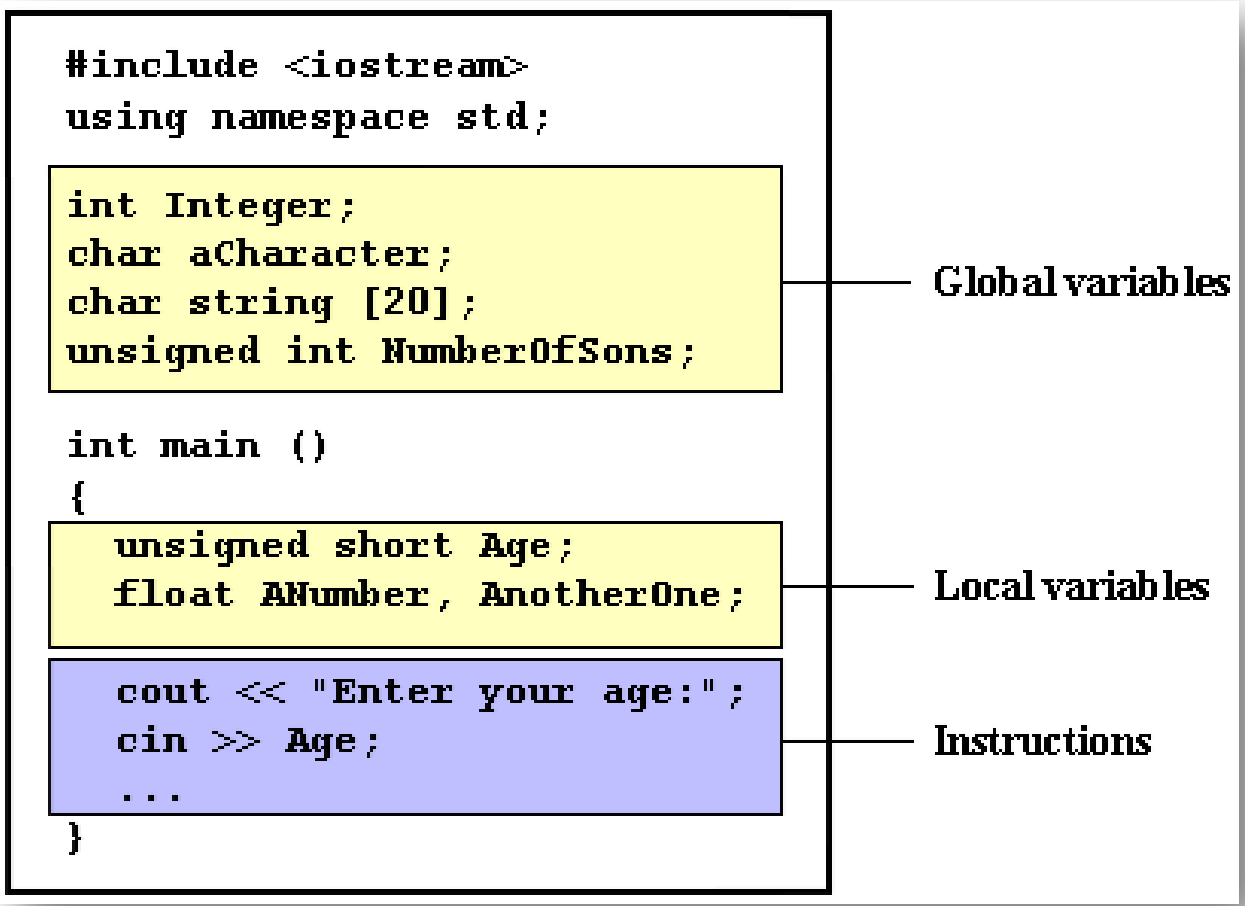
Variável– Local onde um dado específico é guardado. Esse dado pode ser livremente modificado no decorrer do algoritmo (daí o nome variável). Devemos utilizar o nome da variável para acessar seu conteúdo e modificá-lo

Dado – Valor que é armazenado em uma variável.

Tipos de Dados

Nome	Descrição	Tamanho	Intervalo
char	Character or small integer	1byte	signed: -128 to 127 unsigned: 0 to 255
int	Integer.	4bytes	signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295
bool	Boolean value. It can take one of two values: true(any number diff. of zero) or false(0).	1byte	true or false
float	Floating point number.	4bytes	1,2e-38 to 3,4e+38.
double	Double precision floating point number.	8bytes	2,2e-308 to 1,8e+308

Escopo de Variáveis



Inicializar Variáveis

```
1  // initialization of variables
2
3  #include <iostream>
4  using namespace std;
5
6  int main ()
7  {
8    int a=5;               // initial value = 5
9    int b(2);              // initial value = 2
10   int result;            // initial value undetermined
11
12   a = a + 3;
13   result = a - b;
14   cout << result;
15
16   return 0;
17 }
```

Inicializar Variáveis

```
1  // initialization of variables
2
3  #include <iostream>
4  using namespace std;
5
6  int main ()
7  {
8    int a=5;               // initial value = 5
9    int b(2);              // initial value = 2
0    int result;            // initial value undetermined
1
2    a = a + 3;
3    result = a - b;
4    cout << result;
5
6    return 0;
7  }
```

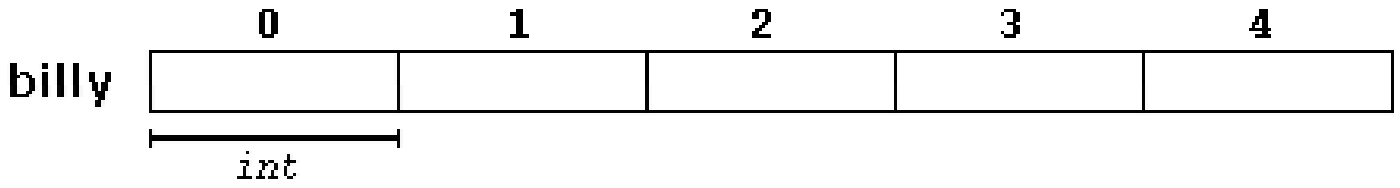
```
6
```

Arrays

Uma array é uma série de elementos do mesmo tipo, colocados em localizações contíguas de memória que podem ser individualmente referenciados por adição de um índice de um identificador único.

Declaração:

```
int billy [5];
```



Arrays

NOTA :

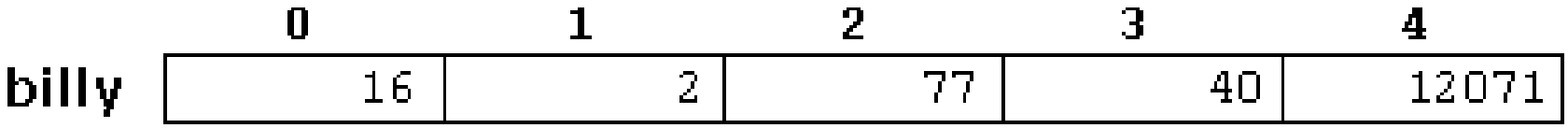
```
int billy [5];
```

O campo de elementos entre colchetes [] que representa o número de elementos do array vai realizar, deve ser um valor constante, já que os arrays são blocos de memória **não-dinâmico** cujo tamanho deve ser determinado antes da execução. A fim de criar arrays com um comprimento variável, é necessário **memória dinâmica**.

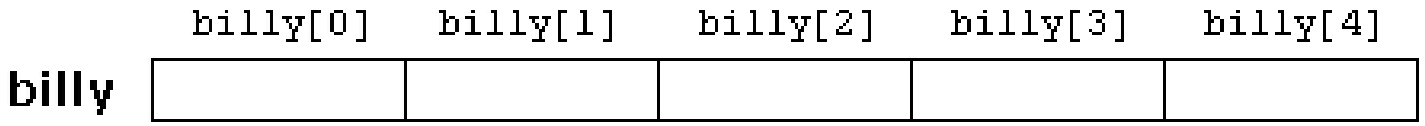
Inicialização de Array

```
int billy [5] = { 16, 2, 77, 40, 12071 };
```

```
int billy [] = { 16, 2, 77, 40, 12071 };
```



Acessando valores de um Array



Atribuição:

```
billy[2] = 75;
```

Acessando:

```
a = billy[2];
```

Mais exemplos:

```
1 billy[0] = a;
2 billy[a] = 75;
3 b = billy [a+2];
4 billy[billy[a]] = billy[2] + 5;
```

Arrays como parâmetros

Os Arrays são passados por referência

A chamada da função na verdade passa o endereço inicial do array.

- Portanto, a função sabe em que possição o array se encontra na memória.

ARRAYS COMO PARÂMETROS

```
1  // arrays as parameters
2  #include <iostream>
3  using namespace std;
4
5  void printarray (int arg[], int length) {
6    for (int n=0; n<length; n++)
7      cout << arg[n] << " ";
8    cout << "\n";
9  }
10
11 int main ()
12 {
13   int firstarray[] = {5, 10, 15};
14   int secondarray[] = {2, 4, 6, 8, 10};
15   printarray (firstarray,3);
16   printarray (secondarray,5);
17   return 0;
18 }
```

Prática

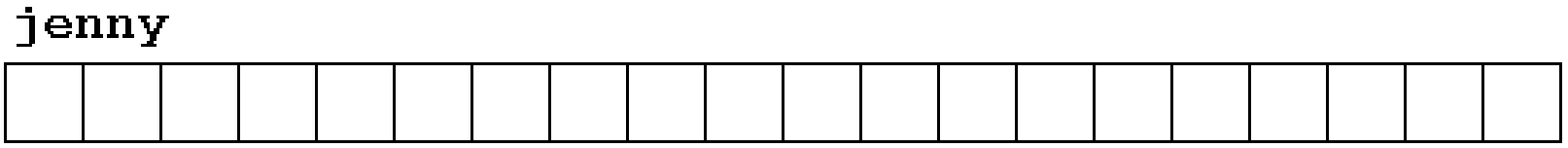
- Implemente o algoritmo anterior, rode. Depois adapte seu código para que o usuário Informe os elementos.
- Implemente um programa em c++ que calcule a soma de todos os elementos informados pelo usuário.

Sequência de Caracteres

Uma cadeia de caracteres é basicamente um array do tipo char.

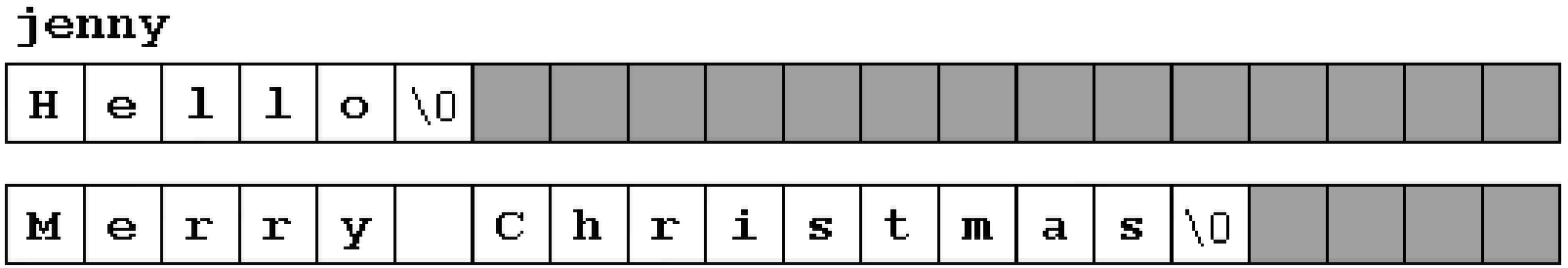
Exemplo:

```
char jenny [20];
```



Sequência de Caracteres

Mesmo com 20 elementos pode usar a quantidade que quiser.



O caracterer ‘\0’ representa o fim da sequência de caracter.

INICIALIZANDO

```
1 char myword [] = { 'H', 'e', 'l', 'l', 'o', '\0' };
2 char myword [] = "Hello";
```

Aspas duplas (") automaticamente acrescenta o caractere nulo. Então as strings entre aspas duplas sempre tem um caractere nulo ('\ 0') automaticamente anexada ao final.

A manipulação dos dados é a mesma do array.

INICIALIZANDO

```
1  // null-terminated sequences of characters
2  #include <iostream>
3  using namespace std;
4
5  int main ()
6  {
7    char question[] = "Please, enter your first name: ";
8    char greeting[] = "Hello, ";
9    char yourname [80];
10   cout << question;
11   cin >> yourname;
12   cout << greeting << yourname << "!";
13   return 0;
14 }
```

```
Please, enter your first name: John
Hello, John!
```

PRÁTICA

- Implemente o programa anterior.
- Implemente um program em c++ que receba uma cadeia de caractere qualquer e indique a quantidade de palavras utilizadas.

Ponteiros

Variáveis são vistos como células de memória que podem ser acessados usando os seus identificadores.

A memória pode ser vista como uma sucessão de células de 1 byte, o qual são numeradas de forma consecutiva, assim como, no interior de um bloco de memória, cada célula tem o mesmo número que o anterior mais um.

Ponteiros

Endereço	Conteúdo							
A013545D	0	1	0	0	1	1	0	1
A013545E	0	1	1	0	1	0	1	1
A013545F	0	1	1	1	1	1	1	1
A0135460	0	0	0	0	0	0	0	0
A0135461	0	1	0	1	1	1	0	1
A0135462	1	0	1	1	1	0	1	1
A0135463	1	0	1	0	0	1	0	1

O conteúdo é binário

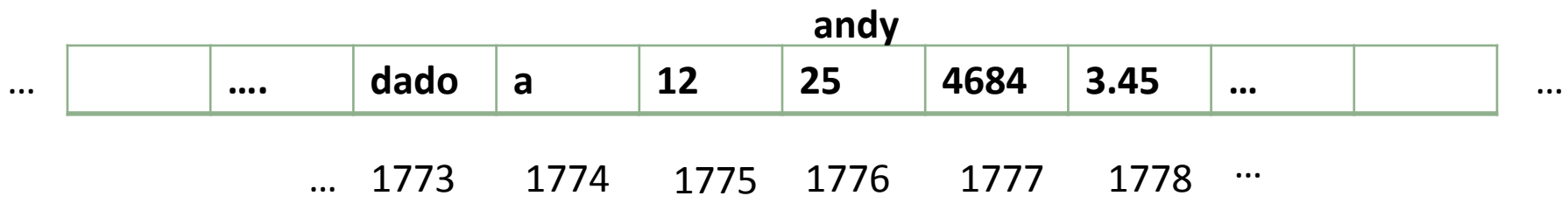
O endereço usa o sistema numérico Hexadecimal.

…		….	dado	a	12	25	4684	3.45	…		…
	…	1773	1774	1775	1776	1777	1778	…			

Reference operator (&)

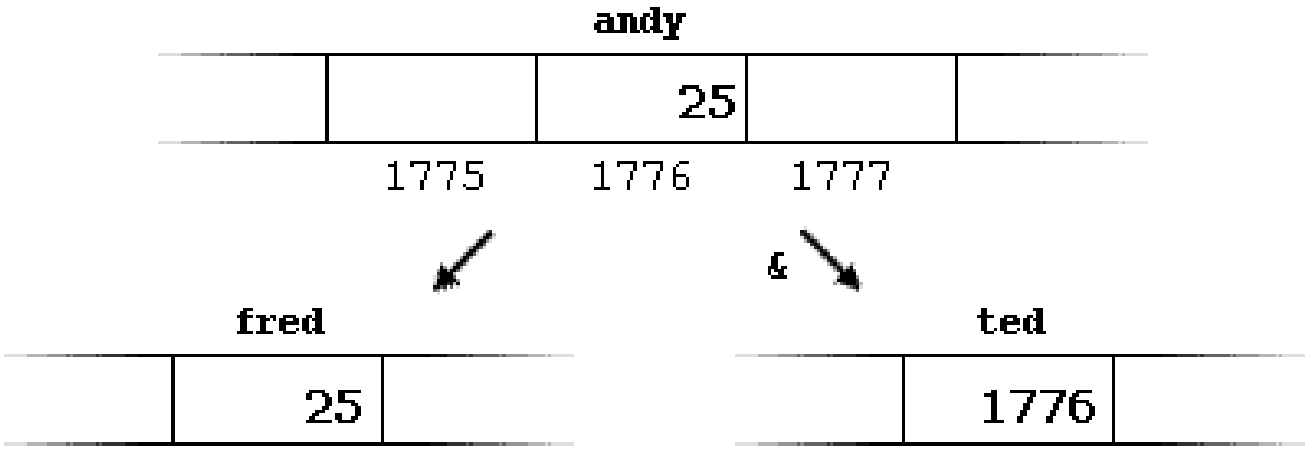
O endereço que localiza uma variável dentro de memória é o que chamamos de uma referência a essa variável.

Esta referência pode ser obtida usando o **sinal &,** conhecido como operador de referência, e que pode ser traduzido literalmente como "endereço de“/”referência de”.



Reference operator (&)

```
1 andy = 25;
2 fred = andy;
3 ted = &andy;
```

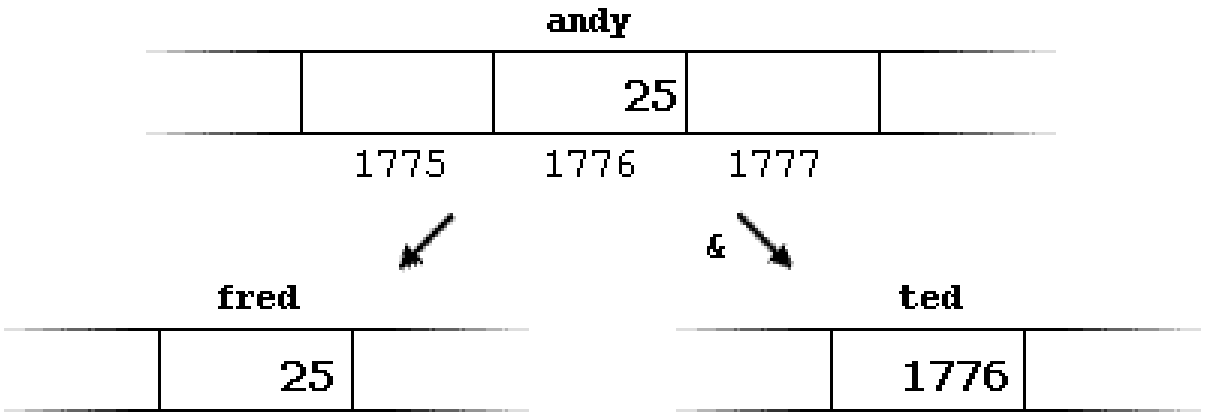


Dereference operator (*)

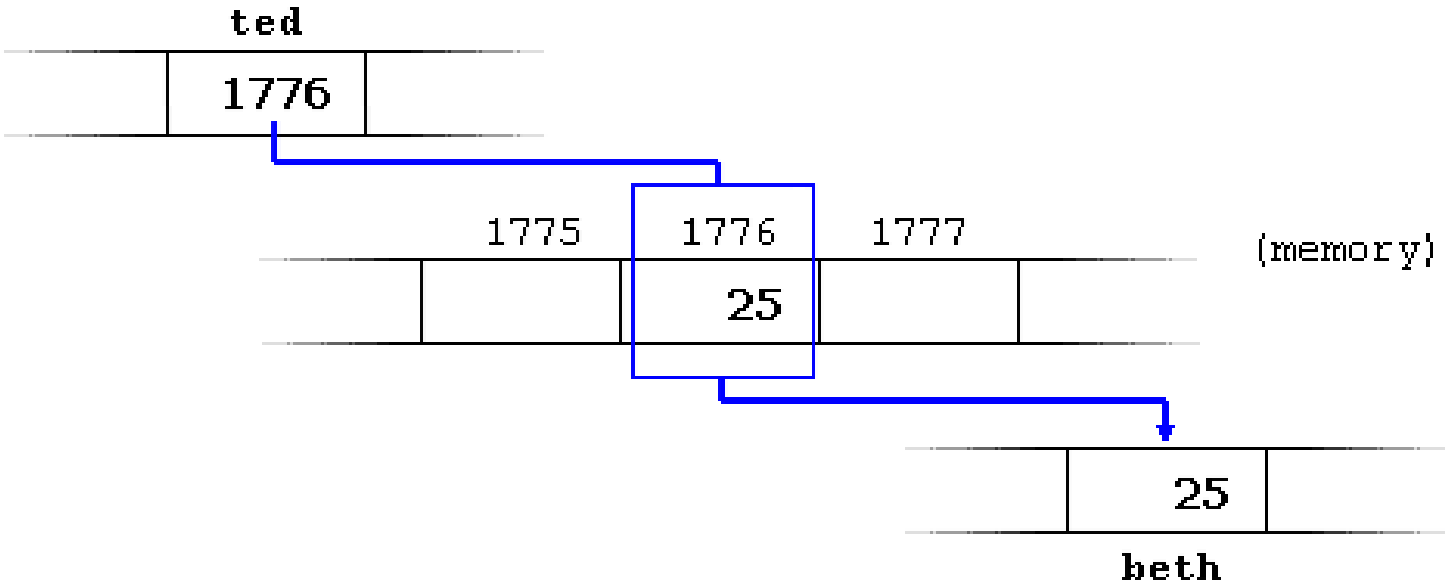
Uma variável que armazena uma referência de outra variável é chamado de ponteiro. Ponteiros são ditos "apontar para" a variável cuja referência eles armazenam.

Usando um ponteiro podemos acessar diretamente o valor armazenado na variável que ele aponta. Para fazer isso, usa-se um asterisco (*), que atua como operador ***dereference*** e que pode ser traduzido literalmente como "valor apontado por".

```
beth = *ted;
```



Dereference operator (*)



Existe alguma diferença?

```
1 beth = ted;
2 beth = *ted;
```

Dereference operator (*)

Tenha em mente que:

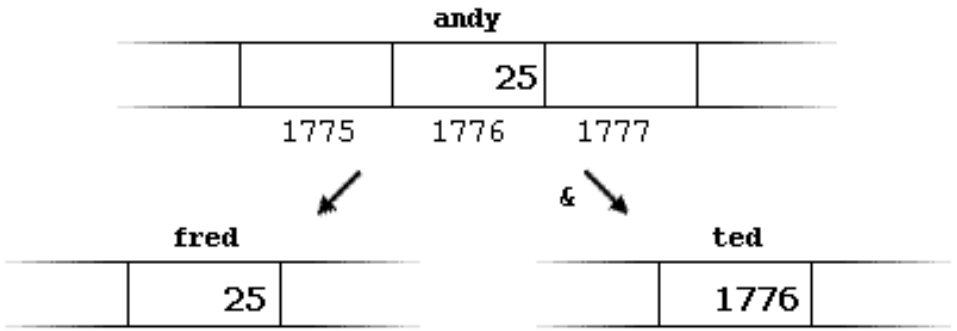
& É o operador de referência e pode ser lido como "endereço de“.

* É o operador *dereference* e pode ser lido como "valor apontado por"

Dereference operator (*)

Assumindo essas duas operações:

```
1 andy = 25;
2 ted = &andy;
```



Logo, todas as igualdades abaixo são verdadeiras?

```
1 andy == 25
2 &andy == 1776
3 ted == 1776
4 *ted == 25
```

```
*ted == andy
```

Declarando variáveis do tipo ponteiro

```
1 int * number;
2 char * character;
3 float * greatnumber;
```

O tipo de dados são diferentes, mas na verdade todos eles são ponteiros e todos eles vão ocupar a mesma quantidade de espaço na memória.

No entanto, os dados a que eles apontam não ocupam o mesmo espaço na memória.

O sinal asterisco (*) usado para declarar um ponteiro só significa que é um ponteiro, e não deve ser confundido com o operador ***dereference***. Eles são duas coisas diferentes representados com o mesmo sinal.

Declarando variáveis do tipo ponteiro

```
1  // my first pointer
2  #include <iostream>
3  using namespace std;
4
5  int main ()
6  {
7    int firstvalue, secondvalue;
8    int * mypointer;
9
10   mypointer = &firstvalue;
11   *mypointer = 10;
12   mypointer = &secondvalue;
13   *mypointer = 20;
14   cout << "firstvalue is " << firstvalue << endl;
15   cout << "secondvalue is " << secondvalue << endl;
16   return 0;
17 }
```

Declarando variáveis do tipo ponteiro

```
1  // more pointers
2  #include <iostream>
3  using namespace std;
4
5  int main ()
6  {
7    int firstvalue = 5, secondvalue = 15;
8    int * p1, * p2;
9
10   p1 = &firstvalue;  // p1 = address of firstvalue
11   p2 = &secondvalue; // p2 = address of secondvalue
12   *p1 = 10;          // value pointed by p1 = 10
13   *p2 = *p1;         // value pointed by p2 = value pointed by p1
14   p1 = p2;           // p1 = p2 (value of pointer is copied)
15   *p1 = 20;          // value pointed by p1 = 20
16
17   cout << "firstvalue is " << firstvalue << endl;
18   cout << "secondvalue is " << secondvalue << endl;
19   return 0;
20 }
```

Inicializar Ponteiros

Mesma inicilização

```
1 int number;
2 int *tommy;
3 tommy = &number;
```

```
1 int number;
2 int *tommy = &number;
```

Ponteiro Nulo:

```
1 int * p;
2 p = 0;      // p has a null pointer value
```

Ponteiro e Arrays

Considere a seguinte declaração:

```
1 int numbers [20];
2 int * p;
```

Fazer isso é válido, já que a variável numbers é um ponteiro para o primeiro Elemento do array.

```
p = numbers;
```

Mas isso não é verdade! Pois, p é somente um ponteiro e não um array.

```
numbers = p;
```

Pode ser tambem considerado como um ponteiro **constante**

Ponteiro e Arrays

```
1  // more pointers
2  #include <iostream>
3  using namespace std;
4
5  int main ()
6  {
7    int numbers[5];
8    int * p;
9    p = numbers;  *p = 10;
10   p++;  *p = 20;
11   p = &numbers[2];  *p = 30;
12   p = numbers + 3;  *p = 40;
13   p = numbers;  *(p+4) = 50;
14   for (int n=0; n<5; n++)
15     cout << numbers[n] << ", ";
16   return 0;
17 }
```

```
10, 20, 30, 40, 50,
```

Ponteiro e Arrays

```
1  // more pointers
2  #include <iostream>
3  using namespace std;
4
5  int main ()
6  {
7    int numbers[5];
8    int * p;
9    p = numbers;  *p = 10;
10   p++;  *p = 20;
11   p = &numbers[2];  *p = 30;
12   p = numbers + 3;  *p = 40;
13   p = numbers;  *(p+4) = 50;
14   for (int n=0; n<5; n++)
15     cout << numbers[n] << ", ";
16   return 0;
17 }
```

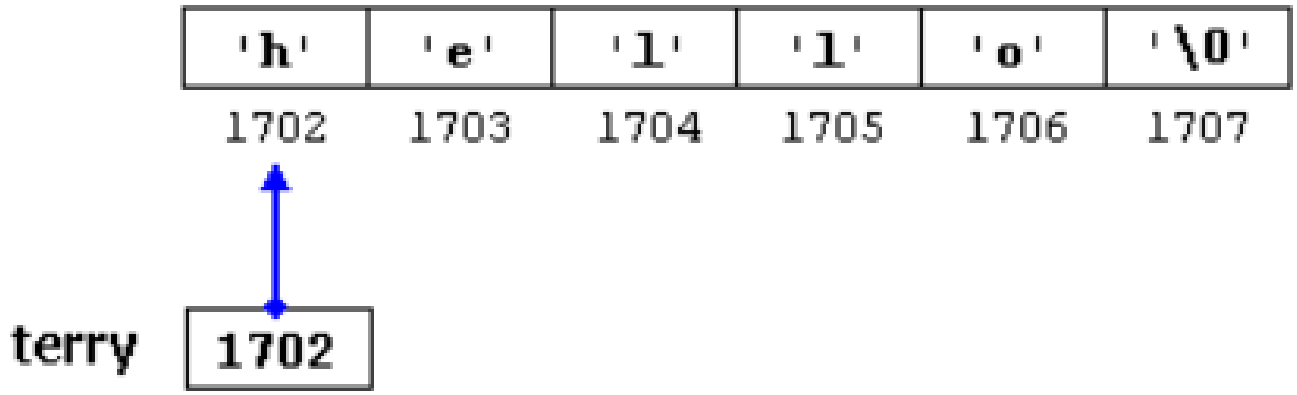
```
10, 20, 30, 40, 50,
```

```
1  a[5] = 0;       // a [offset of 5] = 0
2  *(a+5) = 0;     // pointed by (a+5) = 0
```

São a mesma coisa!!

Ponteiro e Arrays

```
const char * terry = "hello";
```



Alocação dinâmica

http://pt.wikibooks.org/wiki/Programar_em_C%2B%2B/Aloca%C3%A7%C3%A3o_din%C3%A2mica_de_mem%C3%B3ria

Mais informações:

Google.com

Youtube.com

Cplusplus.com

http://pt.wikibooks.org/wiki/Programar_em_C%2B%2B

http://orion.lcg.ufrj.br/C++/curso/#

C++ Como Programar 5ª Edição - Deitel

http://cplusplus.com/reference/cctype/

http://cplusplus.com/reference/cstring/

http://informatica.hsw.uol.com.br/programacao-em-c26.htm

BONS ESTUDOS :)

Algoritmos

Prof. Demétrios Coutinho

INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
RIO GRANDE DO NORTE

Campus Pau dos Ferros
Disciplina de Algoritmos