Caracteres (`char`)

Caracteres (`char`):

- São inteiros de pequeno valor (um byte).
- Os `char` são tratados diferentemente dos demais inteiros na entrada e na saída de dados:
  - Na impressão, ao invés de imprimir o valor do inteiro, imprime o caractere cujo código ASCII é igual ao valor do inteiro.
  - Na leitura, ao invés de atribuir o valor digitado, atribui o código ASCII correspondente ao primeiro caractere digitado.

```
int i;
char c;
i = 50;
c = 50;
cout << "i=" << i << endl;  // Imprime 50
cout << "c=" << c << endl;  // Imprime 2: ASCII('2') == 50
cin >> i;                   // Digite 2
cin >> c;                   // Digite 2
cout << "i=" << i << endl;  // Imprime 2
cout << "c=" << c << endl;  // Imprime 2
i = c;
cout << "i=" << i << endl;  // Imprime 50
```

- As aspas simples `''` são um recurso da linguagem para significar o código ASCII correspondente a um caractere:

```
c = '2';      // Equivale a c = 50
i = 2*'A' + 1 // Equivale a i = 2*65 +1: ASCII('A') == 65
```

Sequências de caracteres

Sequências de caracteres podem ser representadas de três maneiras distintas:

- `string`: classe C++, com alocação dinâmica de memória.

```
#include <string>
string SC;
```

- Array de caracteres (“string” estática em C). Nesse caso, a quantidade de caracteres nunca pode exceder o tamanho máximo que foi alocado na criação do array.

```
char AC[20];
```

- Ponteiro para caracteres (“string” dinâmica em C). Nesse caso, o ponteiro deve sempre apontar para uma região de memória válida, com tamanho igual ou maior do que a quantidade de caracteres que se deseja armazenar.

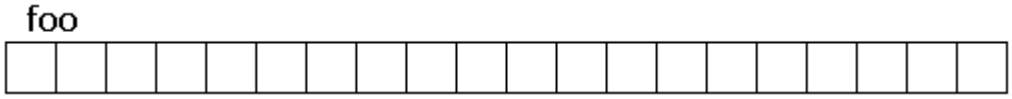
```
char* PC;
```

Arrays de caracteres

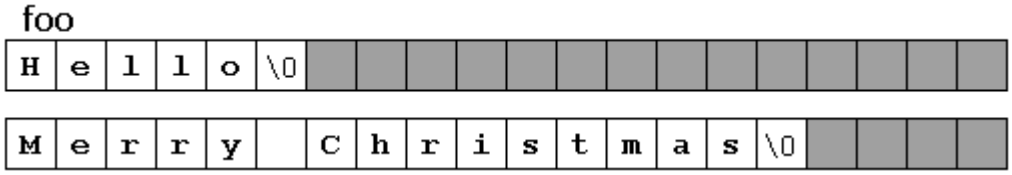
O array de caracteres:

```
char foo[20];
```

é um array de 20 elementos do tipo `char`. Pode ser visualizado como:



Essa capacidade não precisa ser exaurida: o array pode acomodar sequências mais curtas. Por convenção, o fim das sequências de caracteres é assinalado por um caractere especial: o caractere nulo, cujo valor literal é 0 (zero) e pode ser escrito como `'\0'`. Nesse exemplo, o array de 20 elementos do tipo `char` chamado `foo` pode ser representado armazenando as sequências de caracteres “Hello” e “Merry Christmas” como sendo:



Os quadrados em cinza representam `char`’s com valores indeterminados. Em razão da necessidade de acrescentar um caractere nulo ao final da sequência, o array `foo` pode armazenar sequências com até 19 caracteres.

Inicialização de arrays de caracteres

Já que arrays de caracteres são arrays comuns, eles seguem as mesmas regras dos demais arrays. Por exemplo, para inicializar um array de caracteres com uma sequência predeterminada de caracteres, pode-se fazer como com qualquer outro array:

```
char myword[] = {'H','e','l','l','o','\0'};  // Array de 6 chars
```

Mas arrays de elementos do tipo `char` têm outra maneira de serem inicializadas: usando *constantes literais* diretamente. As constantes literais são especificadas na linguagem C/C++ ao se delimitar um texto entre aspas duplas:

```
char myword[] = "Hello";
```

As duas expressões anteriores de inicialização do array `myword` são equivalentes. O caractere nulo é automaticamente adicionado ao se utilizar constantes literais.

É importante ressaltar que os exemplos acima tratam de inicializar um array de caracteres no momento em que ele está sendo declarado, e não de atribuir um valor a ele depois que ele já foi declarado. Os arrays de caracteres, como todos os arrays, não podem ter valores atribuídos a eles, embora cada um dos elementos possa ter um valor atribuído individualmente:

```
char myword[] = "Hello";     // OK
myword = "Bye";              // ERRO DE COMPILAÇÃO
myword[] = "Bye";            // ERRO DE COMPILAÇÃO
myword = {'B','y','e','\0'};// ERRO DE COMPILAÇÃO
myword[0] = 'B';             // OK
myword[1] = 'y';             // OK
myword[2] = 'e';             // OK
myword[3] = '\0';            // OK
strcpy(myword, "Bye");       // ± OK: PERIGOSA – EVITAR USAR
```

Ponteiros para caracteres

Para que os ponteiros para `char` possam ser utilizados para representar sequências de caracteres, eles devem apontar para áreas de memória onde os caracteres possam ser armazena-

dos, e estejam finalizados pelo caractere nulo. Ao contrário dos arrays, podem ter seu valor atribuído, passando a apontar para uma nova área de memória, sendo responsabilidade do programador garantir que a antiga área seja liberada antes da atribuição, caso tenha sido alocada dinamicamente, e que o novo valor do ponteiro aponte para uma sequência de caracteres válida (terminada pelo caractere nulo). Caso o ponteiro aponte para uma constante literal, os valores individuais dos caracteres não podem ser alterados, pois são valores constantes.

```
const char* myword = "Hello"; // OK
char* myword = "Hello";       // ± OK: não proíbe alteração
myword = "Bye";               // OK: aponta para outra constante
myword[0] = 'B';              // ERRO: alteração de constante
myword[1] = 'y';              // ERRO: alteração de constante
myword[2] = 'e';              // ERRO: alteração de constante
myword[3] = '\0';             // ERRO: alteração de constante
strcpy(myword, "Bye");        // ERRO: alteração de constante
```

Entrada/saída de sequências de caracteres

Da mesma forma que o os `char` são tratados diferentemente dos outros inteiros na entrada e saída de dados, os ponteiros (ou arrays) para `char` são tratados diferentemente dos demais ponteiros (ou arrays).

Quando se imprime um ponteiro (ou array), normalmente é impresso o endereço para onde o ponteiro aponta, ou o endereço do primeiro byte da área de memória do array. Já com ponteiros para `char`, vão ser impressos todos os caracteres da sequência de caracteres que se inicia no endereço para o qual o ponteiro aponta, até que seja encontrado o caractere nulo.

Na entrada de dados, normalmente não faz sentido se ler o valor de um ponteiro ou array. Já para ponteiros (ou arrays) para `char`, em algumas situações a linguagem pode entender tal operação como uma leitura de todos os elementos individuais da sequência de caracteres. Essa operação pode conter riscos, pois é necessário garantir previamente que a área de memória associada ao ponteiro/array tem bytes suficientes para conter todos os caracteres que venham a ser digitados pelo usuário. Mais detalhes serão abordados ao se tratar do tópico específico de entrada e saída de dados.

String

A classe `string`, de C++, gerencia toda alocação dinâmica de memória necessária para armazenar sequências de caracteres de tamanho variável. Por essa razão, tem facilidade de uso bem maior e probabilidade de erros bem menor do que os arrays/ponteiros para `char` da linguagem C. Por essa razão, deve ser utilizada sempre que possível.