1.Arrays

Um ***array*** é um grupo de locais da memória relacionados pelo fato de que todos têm o mesmo nome e o mesmo tipo. Para fazer referência a um determinado local ou elemento no ***array***, especificamos o nome do ***array*** e o número da posição daquele elemento no ***array***. Na realidade, os colchetes usados para conter o subscrito de um ***array*** são considerados um operador pela linguagem C. Eles possuem o mesmo nível de precedência que os parênteses.

O primeiro elemento em qualquer ***array*** é o elemento zero. Dessa forma, o primeiro elemento do ***array*** é chamado de ***array[0]***, o segundo elemento do ***array*** é chamado ***array[1]*** e assim por diante. O número de posições contido entre os colchetes é chamado de subscrito. Um subscrito deve ser um inteiro ou uma expressão inteira. Se um programa usar uma expressão inteira como subscrito, a expressão é calculada para determinar o valor do subscrito

Um ***array*** com 5 elementos
Parte esquerda: nome do ***array***
Dentro do colchete: número de posições do elemento dentro do ***array*** C

array[0]	5
array[1]	7
array[2]	9
array[3]	11
array[4]	13

Os ***arrays*** ocupam espaço na memória. O programador especifica o tipo de cada elemento e o número de elementos exigidos por ***array*** de forma que o computador possa reservar a quantidade apropriada de memória. Para dizer ao computador para reservar x elementos para um certo “array” inteiro, é usada a declaração.

```
int array[x];
```

Os elementos de um ***array*** também podem ser inicializados na declaração do ***array*** com um sinal de igual e uma lista de inicializadores separada por vírgulas. Se houver menos inicializadores do que o número de elementos do ***array***, os elementos restantes são inicializados automaticamente com o valor zero, mas é necessário inicializar o primeiro elemento com zero para que os elementos restantes sejam automaticamente zerados.

Se o tamanho do ***array*** for omitido de uma declaração com uma lista de inicializadores, o número de elementos do ***array*** será o número de elementos da lista de inicializadores. A

linguagem C não verifica os limites do ***array*** para evitar que o computador faça referência a um elemento não existente.

Arrays são capazes de conter dados de qualquer tipo. Os ***arrays*** de caracteres possuem variáveis características exclusivas. Um ***array*** de caracteres pode ser inicializado usando uma string literal.

char string1[] = 'primeiro';

Essa ***string*** inicializa os elementos do ***array*** com os valores de cada um dos caracteres da string literal ***primeiro***. O tamanho do ***array string1*** na declaração anterior é determinado pelo compilador, com base no comprimento da ***string***.

É importante notar que string ***primeiro*** contém 8 caracteres mais um caractere especial de término de string, chamado caractere nulo. Dessa forma, o ***array string1*** contém na verdade 6 elementos. A representação da constante do caractere nulo é ***'\0'***. Todas as ***strings*** em C terminam com esse caractere. Um ***array*** de caracteres representado por uma ***string*** deve sempre ser declarado com tamanho suficiente para conter o número de caracteres da ***string*** mais o caractere nulo de terminação.

Os ***arrays*** de caracteres também podem ser inicializados com constantes de caracteres isolados em uma lista de inicializadores.

char string1[] = {'p', 'r', 'i', 'm', 'e', 'i', 'r', 'o'};

É possível também fornecer uma string diretamente a um ***array*** de caracteres a partir do teclado usando ***scanf*** e a especificação de conversão ***%s***.

É responsabilidade do programador assegurar que o ***array*** para o qual a string é lida pode conectar qualquer string que o usuário digite no teclado. A função ***scanf*** lê caracteres do teclado até que o primeiro caractere em branco seja encontrado. Dessa forma, ***scanf*** poderia ir além do final do ***array***.

Podemos aplicar ***static*** à declaração local do ***array*** para que este não seja criado e inicializado sempre que a função for chamada e não seja destruído sempre que a função acabar de ser executada pelo programa. Isso reduz o tempo de execução do programa, particularmente para programas com funções que são chamadas frequentemente e contém ***arrays*** grandes.

2.Passagem de arrays para funções

Para passar um argumento ***array*** a uma função, deve-se especificar o nome do ***array*** sem colocar colchetes.

modifica_array(*array*,*tamanho_array*);

A instrução de chamada de função passa o ***array*** e seu tamanho para a função **modifica_array**. A linguagem C passa automaticamente os ***arrays*** às funções usando chamadas por referências simuladas. As funções chamadas podem modificar os valores dos elementos nos ***arrays*** originais dos locais que fazem as chamadas. O nome do ***array*** é o endereço do primeiro elemento do ***array***. Por ser passado o endereço inicial do ***array***, a função chamada sabe precisamente onde o ***array*** está armazenado. Portanto, quando a função chamada modifica os elementos do ***array*** em seu corpo de função, os elementos reais estão sendo modificados em suas posições originais da memória.

Embora ***arrays*** inteiros sejam passados simultaneamente por meio de chamadas por referência, os elementos individuais dos ***arrays*** são passados por meio de chamadas por valor exatamente como as variáveis simples. Tais porções simples de dados são chamadas de escalares ou quantidades escalares. Para passar um elemento de um ***array*** para uma função, use o nome do elemento do ***array*** com o subscrito, como argumento na chamada da função.

Para uma função receber um ***array*** por meio de uma chamada de função, sua lista de parâmetros deve especificar que um ***array*** será recebido.

void modifica_array(int matrix[], int tamanho);

Podem existir muitas situações em seu programa nas quais não se deve permitir que uma função modifique os elementos de um ***array***. Como os ***arrays*** são sempre passados por chamadas por referência simuladas, as modificações nos valores de um ***array*** são difíceis de controlar. A linguagem C fornece o qualificador especial de tipo ***const*** para evitar a modificação dos valores de um ***array*** em uma função. Isso permite que o programador corrija um programa para que não se tenta modificar o valor dos elementos do ***array***.

3.Introdução a biblioteca de classificação e transformação de strings, <ctype.h>

A biblioteca de classificação de caracteres inclui várias funções que realizam testes úteis e manipulações de dados de caracteres. Cada função recebe um caractere ou EOF como argumento. Normalmente, EOF tem o valor **-1** e algumas arquiteturas de hardware não permitem que valores negativos sejam armazenados em variáveis ***char***. Portanto, as funções de manipulação de caracteres tratam os caracteres como inteiros.

Função	Protótipo	Descrição
isalnum()	**int isalnum** (int *value*);	Verifica se o caractere de entrada é decimal, ou uma letra maiúscula ou minúscula; Valor diferente de 0 é verdadeiro; Valor 0 é falso

isalpha()	**int isalpha** (int *value*);	Verifica se o caractere de entrada é uma letra alfabética; Observe que depende de qual local está se usando. No padrão C, *locale* é apenas quando os retornos de ***isupper*** e ***islower*** são verdadeiros; Um valor diferente de 0 se o caractere for alfabético, caso contrário, 0.
isblank()	**int isblank** (int *value*);	Verifica se o caractere é um caractere em branco. Um caractere em branco é um caractere de espaço, usado para separar palavras. O padrão de C para o caractere em branco é (***'\t'***) ou (***' '***); Um retorno diferente de 0, o retorno é verdadeiro, caso contrário, não
iscntrl()	**int iscntrl** (int *value*);	Verifica se o caractere de entrada é um caractere de controle. Um caractere de controle é aquele que não ocupa uma posição de print no display. Para o padrão ASCII, caracteres de controle são escolhidos entre 0x00 (NULL) e 0x1f (US), também 0x7f (DEL); Um valor de retorno diferente de 0 é verdadeiro, ao contrário, falso
isdigit()	**int isdigit** (int *value*);	Verifica se o caractere inserido é um dígito decimal (0-9); Um valor de retorno diferente de 0 é verdadeiro, caso contrário, falso
isgraph()	**int isgraph** (int *value*);	Verifica se um caractere de entrada é um caractere com representação gráfica
islower()	**int islower** (int *value*);	Verifica se o caractere de entrada é uma letra minúscula. A letra a ser verificada depende de *locale*. Apesar, o retorno sempre é o mesmo; Um valor diferente de 0, significa verdadeiro, caso contrário, falso
isprint()	**int isprint** (int *value*);	Verifica se o primeiro caractere é apresentável. Um caractere apresentável ocupa uma posição de mostragem no display. É o oposto do caractere de controle. Para o padrão ASCII, mostrar caracteres com valor maior do que 0xf1(US), exceto 0xf7(DEL); Um valor de retorno diferente de 0 é verdadeiro, caso contrário, falso
ispunct()	**int ispunct** (int *value*);	Verifica se o caractere de entrada é um caractere de pontuação. O padrão da linguagem C considera caracteres de pontuação como caracteres gráficos que não são alfanuméricos. Outros *locales* podem considerar diferentes seleções de caracteres; Um valor de retorno diferente de 0 é verdadeiro, caso contrário é falso

isspace()	**int isspace** (int *value*);	Verifica se o caractere inserido é um caractere em branco. Uma lista de caracteres em branco pela ASCII está descrito mais abaixo
isupper()	**int isupper** (int *value*);	Verifica se o caractere de entrada é uma letra alfabética maiúscula. O alfabeto varia conforme o *locale*, porém, o retorno sempre tem mesmo valor; Um valor de retorno diferente de 0 é verdadeiro, caso contrário, falso
isxdigit()	**int isxdigit** (int *value*);	Verifica se o caractere inserido é um valor hexadecimal ou não. Um valor retornado diferente de 0 é verdadeiro, caso contrário, falso
tolower()	**int tolower** (int *value*);	Converte um caractere para um caractere minúsculo. Se nenhuma conversão for possível, nenhuma alteração é feita no caractere
toupper()	**int toupper** (int *value*);	Converte um caractere para seu caractere maiúsculo. Se nenhuma conversão for possível, o valor retornado é inalterado

caractere em branco	tabela ASCII	definição
‘ ‘	0x20	space (SCP)
‘\t’	0x09	tab horizontal (TAB)
‘\n’	0x0a	nova linha (LF)
‘\v’	0x0b	tab vertical (VT)
‘\f’	0x0c	feed (FF)
‘\r’	0x0d	carriage return (CR)

Valores ASCII	Caracteres
0x00…0x08	NUL, outros valores de controle
0x09	tab (‘\t’)
0x0A…0x0D	controles de espaço em branco

0x0E…0x1F	outros caracteres de controle
0x20	caractere de espaço (‘ ‘)
0x21…0x2F	! “ # $ % & ‘ () * + , 0 . /
0x30…0x39	0 1 2 3 4 5 6 7 8 9
0x3A…0x40	: ; < = > ? @
0x41…0x46	ABCDEF
0x47…0x5A	GHIJKLMNOPQRSTUVW XYZ
0x5B…0x60	[\]^_´
0x61…0x66	abcdef
0x67…0x7A	ghijklmnopqrstuvwxyz
0x7B…0x7E	{|}~
0x7F	DEL

3.Introdução a biblioteca de manipulação de strings e arrays, <string.h>

Função	Protótipo	Descrição
strncpy()	**char *strncpy**(char **destination*, const char * *source*, size_t *num*);	Copia o primeiro caractere da origem até o destino. Se o final da string de origem for encontrado antes que os caracteres tenham sido copiados, o destino será preenchido com zeros até que um total de caracteres necessários tenha sido gravado nele. Nenhum caractere nulo será anexado implicitamente no final do destino se a origem for maior que a quantidade de destino
strcat()	**char *strcat** (char **destination*, const char **source*);	Acrescenta uma cópia da string de origem à string de destino. O caractere nulo de terminação em destino é substituído pelo primeiro caractere de origem, e um

		caractere nulo é incluído no final da nova string formada pela concatenação’
strcmp()	**int strcmp** (const char **string1*, const char **string2*);	Faz uma comparação entre a *string1* e a *string2*. Um retorno de 0 as strings são iguais; Um retorno maior que 0, o primeiro caractere diferente tem maior valor em 1 Um retorno menor que 0, o primeiro caractere diferente tem menor valor em 1
strlen()	**size_t strlen** (const char **string*);	Retorna o comprimento da string C. O comprimento é determinado pelo caractere nulo de terminação. Uma string C é tão longa quanto o número de caracteres entre o início da string e o caractere nulo de terminação