Manual de C/C++

Versão 1.0

Colabora com a evolução deste manual enviando um e-mail com sugestões, críticas e correções para serafini@cefetes.br.

Colaboradores:
Ernani Ribeiro Filho
José Inacio Serafini

CAPÍTULO

1

LINGUAGEM C/C++

Introdução

A linguagem C é uma linguagem de programação de computadores de uso geral. Foi criada por Dennis Ritchie no Bell Laboratories.

A linguagem C++ pode ser consderada como uma extensão à linguagem C, e também foi desenvolvido no Bell Laboratories por Bjarnes Stroustrup.

Elementos do Programa

Conjunto de caracteres

Um programa fonte pode ser composto de:

- 52 caracteres alfabéticos maiúsculos e minúsculos.

```
ABCDEFGHIJKLMNOPQRSTUVXWYZ
abcdefghijklmnopqrstuvxywz
```

- 10 digitos de decimais

```
0123456789
```

- dos símbolos “espaço”, “tabulação horizontal”, “tabulação vertical” e “avança formulário-form feed”
- 29 simbolos gráficos

```
!#%^&*()-_+=~{}\|;:'"[],.<>/?
```

Comentários

Temos dois tipos de comentários:

- Os comentários são colocados entre /* e */ e não são considerados pelo compilador. Podem ocupar mais de uma linha e não podem ser colocados dentro de outros comentários.
- Todo texto após // até o final da linha. Só pode ocupar uma linha.

Exemplo:

```
main() /* funcao obrigatoria */
{
  printf("oi"); // imprime "oi".
}
```

Identificadores

São nomes usados para se fazer referência a variáveis, funções, rótulos e vários outros objetos definidos pelo usuário. O primeiro caractere deve ser uma letra ou um sublinhado. Os 32 primeiros caracteres de um identificador são significativos (case sensitive), ou seja, as letras maiúsculas diferem das minúsculas.

```
int x; /*é diferente de int X;*/
```

Variáveis

Todas as variáveis em C/C++ devem ser declaradas ou definidas antes de serem utilizadas.

Na **declaração** da variável indicamos o seu tipo. Uma **definição** ocorre quando declaramos uma variável e atribuimos algum valor inicial a ela.

Nomes de variáveis em C/C++

Os nomes de variáveis em C/C++ podem conter letras, números e caracteres de underscore (_).

Como C/C++ distingue o tipo de letra as letras maiusculas são consideradas diferentes das minusculas.

As vezes é interessante utilizar o caractere underscore (_) ou uma mistura de maiusculas e minusculas para melhoras a legibilidade.

Exemplo: `dia_da_semana` ou `diaDaSemana`

Podemos declarar mais de uma variável em uma mesma declaração, baste que separemos as declarações por uma virgula.

Exemplo: `int a, b, c d;`

A palavra reservada `const` permite definir "variáveis" com valores constantes, isto é, valores que não podem ser modificados

Em ANSI C todas as declarações de variáveis devem ser feitas no início do programa ou função. Se for necessario fazer declarações adicionais o programador deve incluir a definição no início do programa. Todas as declarações devem ser feitas antes da execução de qualquer comando.

Em C++ as declarações de variáveis podem ser feitas em qualquer lugar do programa. Esta característica permite que o programador possa declarar suas variáveis próximas do lugar onde ela vai ser utilizada.

Sintaxe: `nomeTipo nomeVariavel1, ... nomeVariaveln;`

Podemos atribuir valores as variáveis:

Exemplo: `int a = 1;`

Existem três tipos básicos de variáveis:

- locais,
- estáticas,
- globais.

Variáveis locais

As variáveis que são declaradas dentro de uma função são chamadas de locais.

Na realidade toda variável declarada entre um bloco { } podem ser referenciadas apenas dentro deste bloco. Elas existem apenas durante a execução do bloco de código no qual estão declaradas. O armazenamento de variáveis locais por default é na pilha, assim sendo uma região dinâmica.

Exemplo:
```
void linha;

main(){
```

Variáveis Estáticas

Funcionam de forma parecida com as variáveis globais, conservando o valor durante a execução de diferentes funções do programa. No entanto só são reconhecidas na função onde estão declaradas. São muitos utilizadas para inicializar vetores.

Exemplo:
```
main()
{
  int i;
  static int x[10]={0,1,2,3,4,5,6,7,8,9};
  for(i=0;i<10;i++)
  printf("%d");
}
```

Variáveis Globais

São conhecidas por todo programa e podem ser usadas em qualquer parte do código. Permanecem com seu valor durante toda execução do programa. Deve ser declarada fora de qualquer função e até mesmo antes da declaração da função main.

Fica numa região fixa da memória própria para esse fim.

Exemplo:
```
void func1(),func2();
int cont;
main()
{
  cont=100;
  func1();
}
void func1()
{
  int temp;
  temp=cont;
  func2();
  printf("cont é = %d",cont);
}
void func2()
{
  int cont;
  for(cont=1;cont<10;cont++) printf(".");
}
```

Palavras Reservadas

As palavras reservadas ou palavras chave não podem ser utilizadas como indetificadores, pois possuem um significado para o compilador.

Tabela 0.1 Palavras Reservadas em C/C++

asm	auto	break	bool	case
catch	char	class	const	cons_char
continue	default	delete	do	double
else	enum	extern	float	for
friend	goto	if	inline	int
long	new	operator	private	protected
public	register	return	short	signed
sizeof	static	struct	switch	template
this	throw	try	typedef	union
unsigned	virtual	void	volatile	while
dynamic_cast	false	mutable	namespace	static_cast
true	typeid	using	wchar_t	reinterpret_cast

Constantes

As constantes são caracterizadas por um valor e um tipo.

C/C++ permite criar constantes com os diversos tipos de dados definidos na linguagem.

Inteiras

Podem ser:

- Decimais: são números decimais sem ponto (virgula). Exemplo: 123, 1, 10, 0, -1, etc.
- Octais: iniciam sempre com um zero. Exemplo: 0123, 01, 010, etc
- Hexadecimais: iniciam com zero-x. Exemplo: 0x123, 0x1, 0x10, etc
- Inteiras Longas: indicadas com um l ou L. Exemplo: com sinal: 123L, ou sem sinal 123UL

Ponto flutuante

São valores numéricos com virgula decimal (ou ponto decimal). Podem ter ou ponto decimal ou expoente. Exemplo: 0., 1e2, 3.141592

Podemos utilizar o sufixo f ou F para indicar float, ou l ou L para indicar tipo long. Exemplo: 1.0f, 1.0L

Constante de caracteres

- Cadeias de caracteres: “uma cadeia definida entre aspas duplas”.

Tipos de Dados e Declarações

Inteiros

Representam números inteiros, isto é numeros sem a virgula decimal.

Tabela 0.2 Tipo Inteiro

Tipo de dado	Tamanho em Bytes	Tamanho em Bits	Menor Valor	Maior Valor
signed char	1	8	-128	127
unsigned char	1	8	0	255
signed short	2	16	-32768	32767
unsigned short	2	16	0	65535
signed int	2	16	-32768	32767
unsigned int	2	1	0	65535
signed long	4	32	-2147483648	2147483647
unsigned long	4	32	0	4294967295

Ponto Flutuante

Tipos Reais - Ponto Flutuante

Representam número reais, isto é, numeros com virgula (ponto) decimal.

Tabela 0.3 Tipo ponto flutuante

Tipo de Dado	Tamanho em Bytes	Tamanho em Bits	Menor Valor	Maior Valor
float	4	32	-3.4E-38	3.4E+38
double	8	64	-1.7E-308	1.7E+308
long double	10	80	-3.4E-4932	1.1E+4932

Arrays

A array é um tipo de dado usado para representar uma certa quantidade de variáveis do mesmo tipo e que são referenciados pelo mesmo nome. Consiste em locações contíguas de memória. O endereço mais baixo corresponde ao primeiro elemento.

Array unidimensional

Sintaxe: `tipo nome[tamanho];`

Os arrays tem 0 como índice do primeiro elemento, portanto sendo declarada uma matriz de inteiros de 10 elementos, o índice varia de 0 a 9.

Exemplo:

```
main(){
  int x[10];
  int t;
  for(t=0;t<10;t++)
  {
    x[t]=t*2;
    printf(" x[%d]=%d\n”, t, x[t]);
  }
}
```

Exemplo:

```
main()
{
```

```
  int notas[5],i,soma;
  for(i=0;i<5;i++)
  {
    printf("Digite a nota do aluno %d: ",i);
    scanf("%d",&notas[i]);
  }
  soma=0;
  for(i=0;i<5;i++)
    soma=soma+notas[i];
  printf("Media das notas: %d.",soma/5);
}
```

Arrays Multidimensionais

Sintaxe: `tipo nome[tamanho][tamanho] ...;`

Funciona como um array de uma dimensão (vetor), porém utilizamos mais de um indice para acessar os elementos.

Exemplo:

```
main()
{
  int x[10][10];
  int t,p;
  for(t=0,p=0;t<10;t++,p++)
  {
    x[t][p]=t*p;
    printf(" x(%d,%d)=%d\n", t, p, x[t][p]);
  }
}
```

Arrays Estáticos

Os vetores de dados podem ser inicializados como os dados de tipos simples, mas somente como variáveis globais. Quando for inicializar uma matriz local sua classe deve ser static.

Exemplo:

```
main()
```

```
{
  int i;
  static int x[10]={0,1,2,3,4,5,6,7,8,9};
  for(i=0;i<10;i++)
    printf(" x[%d] = %d\n", i, x[i]);
}
```

Limites dos Arrays

A verificação de limites não é feita pela linguagem, nem mensagem de erros são enviadas, o programa tem que testar os limites das matrizes.

Exemplo:

```
main()
{
int erro[10],i;
for(i=0;i<100;i++)
{
erro[i]=1;
printf("%d

}
}
```

Ponteiros

Sintaxe: `tipo *nomevar;`

É uma variável que contém o endereço de outra variável. Os ponteiros são utilizados para alocação dinâmica, podendo substituir matrizes com mais eficiência.

Também fornecem a maneira pelas quais funções podem modificar os argumentos chamados, como veremos no capítulo de funções.

Declarando Ponteiros

Se uma variável irá conter um ponteiro, então ela deve ser declarada como tal:

```
int x,*px;
```

```
px=&x; /*a variável px aponta para x */
```

Se quisermos utilizar o conteúdo da variável para qual o ponteiro aponta:

```
y=*px;
```

O que é a mesma coisa que:

```
y=x;
```

Manipulação de Ponteiros

Como os pointers são variáveis, eles podem ser manipulados como as variáveis comuns. Se py é um outro ponteiro para um inteiro então podemos fazer a declaração:

```
py = px;
```

Exemplo:

```
main(){
  int x,*px,*py;
  x=9;
  px=&x;
  py=px;
  printf("x= %d");
  printf("&x= %d");
  printf("px= %d");
  printf("*px= %d");
  printf("*px= %d");
}
```

Expressões com Ponteiros

Os ponteiros podem aparecer em expressões, se px aponta para um inteiro x, então *px pode ser utilizado em qualquer lugar que x seria.

O operador * tem maior precedência que as operações aritméticas, assim a expressão abaixo pega o conteúdo do endereço que px aponta e soma 1 ao seu conteúdo.

```
y=*px+1;
```

No próximo caso somente o ponteiro será incrementado e o conteúdo da próxima posição da memória será atribuído a y:

```
y=*(px+1);
```

Os incrementos e decrementos dos endereços podem ser realizados com os operadores ++ e --, que possuem procedência sobre o * e operações matemáticas e são avaliados da direita para a esquerda:

```
*px++; /* sob uma posição na memória */
*(px--); /* mesma coisa de *px-- */
```

No exemplo abaixo os parênteses são necessários, pois sem eles px seria incrementado em vez do conteúdo que é apontado, porque os operadores * e ++ são avaliados da direita para esquerda.

```
(*px)++ /* equivale a x=x+1; ou *px+=1 */
```

Exemplo:

```
main()
{
  int x,*px;
  x=1;
  px=&x;
  printf("x= %d");
  printf("px= %u");
  printf("*px+1= %d");
  printf("px= %u");
  printf("*px= %d");
  printf("*px+=1= %d");
  printf("px= %u");
  printf("(*px)++= %d");
  printf("px= %u");
  printf("*(px++)= %d");
  printf("px= %u");
  printf("*px++-= %d");
  printf("px= %u");
}
```

Ponteiros para ponteiros

Um ponteiro para um ponteiro é uma forma de indicação múltipla. Num ponteiro normal, o valor do ponteiro é o valor do endereço da variável que contém o valor desejado. Nesse caso o primeiro ponteiro contém o

endereço do segundo, que aponta para a variável que contém o valor desejado.

```
float **balanço;
```

balanço é um ponteiro para um ponteiro float.

Exemplo:

```
main()
{
  int x,*p,**q;
  x=10;
  p=&x;
  q=&p;
  printf("%d",**q);
}
```

Problemas com ponteiros

O erro chamado de ponteiro perdido é um dos mais difíceis de se encontrar, pois a cada vez que a operação com o ponteiro é utilizada, poderá estar sendo lido ou gravado em posições desconhecidas da memória. Isso pode acarretar em sobreposições sobre áreas de dados ou mesmo área do programa na memória.

```
int,*p;
x=10;
*p=x;
```

Estamos atribuindo o valor 10 a uma localização desconhecida de memória. A consequência desta atribuição é imprevisível.

Enumerador - `enum`

O tipo de dado enumerado é uma lista ordenada de elementos composto de constantes inteiras. A menos que se indique o contrário o primeiro membro de um conjunto enumerado de valores recebe o valor 0, porém podemos especificar outros valores caso necessário.

Utilizamos a palavra reservada enum para declarar um tipo de dado enumerado ou enumeração.

Sintaxe:

```
enum <nome>
{
```

```
  <lista de simbolos>
}
```

onde <nome> é o nome da variável declarada enumerada e <lista de simbolos> é uma lista de tipos enumerados aos quais são atribuídos valores quando se declara a variável enumeradas e que podem ter um valor de inicialização.

Podemos utilizar o nome de uma enumeração para declarar uma variável deste tipo.

Exemplo:
```
enum cor { vermelho, verde, azul };
...
cor janela = vermelho;
```

Exemplo:
```
enum diaSemana {segunda, terca, quarta, quinta, sexta,
sabado, domingo};
// segunda = 0
// terca = 1
// ...
// se fizermos:
sexta = 10;
// então sabado = 12 e domingo = 13
// ...
diaServico = sabado
if (diaServico >= sexta)
  printf("Hone não tem servico...");
// ...
```

Estruturas - `struct`

Ao manusearmos dados muitas vezes deparamos com informações que não são fáceis de armazenar em variáveis escalares como são os tipos inteiros e pontos flutuantes, mas na verdade são conjuntos de coisas. Este tipo de dados são compostos com vários dos tipos básicos do C. As estruturas permitem uma organização dos dados dividida em campos e registros.

Uma estrutura é um tipo de dado composto que contém uma coleção de elementos de tipos de dados diferentes combinados em uma única

construção da linguagem. Cada elemento da coleção se chama membro e pode ser uma variável de um tipo diferente.

Uma estrutura representa um novo tipo de dado em C/C++.

Sintaxe:
```
struct <nome_da_estrutura>
{
  <membros>
}
```

Exemplo:
```
struct ponto
{
  int x;
  int y;
}
...
  struct ponto p1;// estilo C
  ponto p2;      // estilo C++
...
  p2.x = 10;     // atribui 10 para x de p2
  p2.y = p2.x + 10;// atribui 20 para y de p2
...
```

Exemplo:
```
struct lapis {
  int dureza;
  char fabricante;
  int numero;
};
main()
{
  int i;
  struct lapis p[3];
  p[0].dureza=2;
  p[0].fabricante='F';
  p[0].numero=482;
  p[1].dureza=0;
```

```
  p[1].fabricante='G';
  p[1].numero=33;
  p[2].dureza=3;
  p[2].fabricante='E';
  p[2].numero=107;
  printf("Dureza Fabricante Numero");
  for(i=0;i<3;i+ +)
    printf("%d%c%d");
}
```

Como ocorre com as variáveis, as estruturas também podem ser referenciadas por ponteiros. Assim, definindo-se por exemplo o ponteiro ***p** para a estrutura acima (lapis), pode-se usar a sintaxe **(*p).dureza**. Porém, para referenciar o ponteiro há ainda outra sintaxe, através do operador **->** , como por exemplo, **p->dureza**.

Uniões - `union`

Uma união (union) é uma variável que pode armazenar objetos de tipos e tamanhos diferentes.

Uma união pode armazenar tipos de dados diferentes mas só pode armazenar um de cada vez, ao contrário de uma estrutura.

Sintaxe:

```
union <nome>
{
  <membros>;
}
```

Exemplo:

```
union valor
{
  int v1;
  float v2;
  double v3;
}
```

C++ adimite um tipo de união chamado união anônima, que declara um conjunto de membros que compartilham o mesmo endereço na memória. Para acessar uma união anônima utilizamos os nomes das variáveis.

Exemplo:
```
union
{
  double v1;
  double v2;
}
```

Funções

Tipo void

nomes Typedef

Operadores e Expressões

Operador de Atribuição

O operador de atribuição em C é o sinal de igual "=". O operador de atribuição = faz com que o valor situado à direita seja atribuído a variável a esquerda. A atribuição pode ocorrer como parte de uma expressão de atribuição e as conversões são feitas implicitamente.

Ao contrário de outras linguagens, o operador de atribuição pode ser utilizado em expressões que também envolvem outros operadores.

```
a = b + 1; // atribui b+1 a variavel a
```

C++ permite atribuição múltipla:

```
a = b + (c=10);
```

é equivalente a:

```
c = 10;
a = b + c;
```

O compilador pode gerar código mais eficiente recorrendo a operadores de atribuição compostos; cada operador composto reduz a expressão do tipo `a = a operação b` em `a operação= b`.

C++ possui operadores de atribuição que combinam operadores de atribuição e com operadores aritméticos e lógicos. Veja a tabela abaixo:

Tabela 0.4 Operadores de atribuição

Operador	Descrição	Exemplo
=	Atribução	a=b;
+=	a=a+b	a+=b;
-=	a=a-b	a-=b;
*=	a=a*b	a*=b;
/=	a=a/b	a/=b;
%=	a=a%b	a%=b;
<<=	a=a<<b	a<<=b;
>>=	a=a>>b;	a>>=b;
&=	a=a&b	a&=b;
^=	a=a^b	a^=b;
|=	a=a|b	a|=b;

Operadores Aritméticos

Os operadores *, /, + e - funcionam como na maioria das linguagens, o operador % indica o resto de uma divisão inteira.

```
i+=2; -> i=i+2;
x*=y+1; -> x=x*(y+1);
d-=3; -> d=d-3;
```

Exemplo:

```
main()
{
  int x,y; x=10; y=3;
  printf("%d");
  printf("%d");
}
```

Operadores Relacionais e Lógicos

Os operadores relacionais permitem descobrir relações entre objetos tais como a é menor que b, ou a é menor ou igual a b, etc.

Os operadores logicos permitem realizar operações booleanas entre objetos ou relações.

Em C/C++ o zero (0) é sempre **FALSO**, enquanto que **VERDADEIRO** é qualquer valor **diferente de zero**.

No entanto as expressões que usam operadores de relação e lógicos retornam 0 para falso e 1 para verdadeiro.

Tanto os operadores de relação como os lógicos tem precedência menor que os operadores aritméticos. As operações de avaliação sempre produzem um resultado 0 ou 1.

Tabela 0.5 Operadores Relacionais

Operador	Significado
>	maior que
>=	maior ou igual
<	menor que
<=	menor ou igual
==	igual a
!=	não igual (diferente)

Tabela 0.6 Operadores Lógicos

Operador	Significado
&&	AND
||	OR
!	NOT

Regras Práticas

Os operadores lógicos e relacionais atuam sobre valores lógicos: o valor falso pode ser tanto o zero quanto o ponteito nulo, ou ainda o 0.0; o valor verdadeior pode ser qualquer valor diferente de zero

Veja os exemplos de resultados de expressões abaixo:

Expressões	Resultado
x > y	se x maior que y retorna 1 senão retorna 0
x >= y	se x é maior ou igual a y retorna 1 senão retorna 0

Expressões	Resultado
x < y	se x é menor que y retorna 1 senão retorna 0
x <= y	se x é menor ou igual que y retorna 1 senão retorna 0
x == y	se x é igual a y então retorna 1 senão retorna 0
x |= y	se x é diferente de y então retorna 1 senão retorna 0
!x	se x é zero então retorna 1 senão retorna 0
x || y	se x e y são 0 então retorna 1 senão retorna 0

Exemplo:

```
main()
{
  int i,j;
  printf("digite dois números: ");
  scanf("%d%d",&i,&j);
  printf("%d == %d é %d");
  printf("%d != %d é %d");
  printf("%d <= %d é %d");
  printf("%d >= %d é %d");
  printf("%d < %d é %d");
  printf("%d > %d é %d");
}
```

Exemplo:

```
main()
{
  int x=2,y=3,produto;
  if ((produto=x*y)>0)
    printf("é maior");
}
```

Avaliação em Curto-circuito

C/C++ permite reduzir o tempo de avaliação das operações lógicas fazendo a avaliação das expressões ser interrompida quando alguns dos operandos tomam valores específicos.

Operações Lógicas AND (&&): se na expressão `expr1 && expr2`, `expr1` toma o valor zero (falso) a operação logica AND será sempre zero qualquer que seja o valor de `expr2`. Portanto `expr2` não será avaliada.

Operações Logicas OR (||): se na expressão `expr1 || expr2`, `expr1` toma o valor diferente de zero (falso) a operação logica OR será sempre igual a 1 qualquer que seja o valor de `expr2`. Portanto `expr2` não será avaliada.

Operadores de Incremento e Decremento

O C fornece operadores diferentes para incrementar variáveis. O operador de incremento soma 1 ao seu operando, e o decremento subtrai 1. O aspecto não usual desta notação é que podem ser usado como operadores pré-fixo(++x) ou pós-fixo(x++).

++x incrementa x antes de utilizar o seu valor.

x++ incrementa x depois de ser utilizado.

Exemplo:
```
a = b++;
```

é equivalente a

```
a = b;
b = b + 1;
```

Exemplo:
```
main()
{
  int x=0;
  printf("x= %d");
  printf("x= %d");
  printf("x= %d");
  printf("x= %d");
}
```

Operadores de Bits

C/C++ possui operadores de manipulação de bits bem como operadores de atribuição de manipulação de bits.

Tabela 0.7 Operadores de manipulação de bits

Operador	Descrição	Exemplo
&	AND bit a bit	x & 256
|	OR bit a bit	x | 128

Tabela 0.7 Operadores de manipulação de bits

Operador	Descrição	Exemplo
^	XOR bit a bit	x ^12
~	NOT bit a bit	~x
<<	desloca a esquerda	x << 3
>>	desloca a direita	x >> 3
&=	x = x & y	
!=	x = x | y	
^=	x = x ^ y	
<<=	x = x << y	
>>=	x = x >> y	

Operador de Endereço

& antes de uma variável significa "endereço da variável".

Exemplo: &a -> endereço na memória da variável a.

Operador de Conteúdo de Endereço

* antes de uma variável significa "o conteúdo do endereço armazenado na variável".

Exemplo: *a -> conteúdo da memória cujo endereço esta armazenado na variável a.

Comandos

Regras Gerais

Comandos simples e compostos

Um **comando simples** é qualquer expressão válida e que termina com um ponto-e-virgula (;).

Exemplo:
```
i++; // ok.
a+b; // comando legal mas não faz nada!
a = b + (c = 2); // equivalente a c=2; e a=b+c;
```

Um **comando composto** é uma série de comandos entre chaves { ... }.

Exemplo:
```
{
  t = a;
  a = b;
  b = t;
}
```

Podemos ter qualquer tipo de comando nos comandos compostos.

Um comando composto pode ser sempre utilizado no lugar de um comando simples.

Comando Condicional

if

Sintaxe:
```
if (expressão) {
  <sequencia de comandos 1>
}
else {
  <sequencia de comandos 2>
}
<comando 3>
```

Se a expressão for verdadeira (diferente de zero) então a <sequencia de comandos 1> será executada e em seguida será executado o <comando 3>.

Caso contrário (a expressão fofr igual a zero) será executada a <sequencia de comandos 2> e em seguida será executado o <comando 3>.

Podemos ter uma instrução if sem a clausula else:

```
if (condição) {
  <sequencia de comandos 1>
}
```

Exemplo:
```
main()
{
  int a,b;
  printf("digite dois números:");
```

```
  scanf("%d%d",&a,&b);
  if (b) printf("%d / %d = %d", a, b, a/b);
  else printf("Erro: divisão por zero");
}
```

if-else-if

Podemos combinar vários ifs. Construindo estruturas de ifs "aninhados" como:

```
if (expressão1)
  comando1;
else if (expressão2)
  comando2;
...
else if (expressãoN)
  comandoN;
```

Exemplo:

```
#include <stdlib.h>
#include <time.h>
main(){
  int num,segredo;
  srand(time(NULL));
  segredo=rand()/100;
  printf("Qual e o numero: ");
  scanf("%d",&num);
  if (segredo==num)
    printf("Acertou!");
  else if (segredo<num)
    printf("Errado, muito alto!");
  else printf("Errado, muito baixo!");
}
```

switch

O comando switch permite realizar decisões de alternativas multiplas.

Sintaxe: `switch(expressão)`

```
{
case valor1: <sequência de comandos 1>; break;
case valor2: <sequência de comandos 2>; break;
...
case valorN: <sequencia de comandos N>; break;
default:sequência de comandos
}
```

O comando switch requer que expressão tenha um valor inteiro. Este valor pode ser uma constante, uma variável, uma chamada de função ou uma expressão. O valor de valor1 ... valorN tem que ser constantes.

Durante a execução a expressão será avaliada e se seu valor coincide com algum valor de valor1 até valorN será executado a sequencia de comando correspondente.

Se nenhuma coincidência for encontrada será executado a sequencia de comandos após o comando default. O comando default é opcional.

A sequência de comandos é executada até que o comando break seja encontrado.

Exemplo:

```
main()
{
  char x;
  printf("1. inclusão"
  printf("2. alteração"
  printf("3. exclusão
  printf(" Digite sua opção:");
  x=getchar();
  switch(x)
  {
    case '1':
      printf("escolheu inclusão"
      break;
    case '2':
      printf("escolheu alteração");
      break;
```

```
    case '3':
      printf("escolheu exclusão");
      break;
    default:
      printf("opção inválida");
  }
}
```

Expressão condicional

Uma expressão condicional é uma simplificação do comando if...else.

Sintaxe: `<condição> ? <expressão1> : <expressão2>;`

É uma maneira compacta de expressar um if-else. É equivalente a:

```
if (condição) {
  <expressão1>
}
else
  <expressão2>
}
```

Exemplo:

```
main()
{
  int x,y,max;
  printf("Entre com dois números: ");
  scanf(%d,%d,&x,&y);
  max = (x>y) ? x: y;
  printf("max= %d");
}
```

Comandos de Repetição - Loop

for

O comando for repete a execução um número determinado de vezes.

Sintaxe: `for(inicialização; condição; incremento) {`

```
  <comandos>;
}
```

O comando for é de alguma maneira encontrado em todas linguagens procedurais de programação. Em sua forma mais simples, a incialização é um comando de atribuição que o compilador usa para estabelecer a variável de controle do loop. A condição é uma expressão de relação que testa a variável de controle do loop contra algum valor para determinar quando o loop terminará. O incremento define a maneira como a variável de controle do loop será alterada cada vez que o computador repetir o loop.

Exemplo:

```
main()
{
  int x;
  for(x=1; x<100; x++) {
    printf("%d");
  }
}
```

Exemplo:

```
main()
{
  int x,y;
  for (x=0,y=0; x+y<100; ++x,++y) {
    printf("%d ",x+y);
  }
}
```

Um uso interessante para o for é o loop infinito, como nenhuma das três definições são obrigatórias, podemos deixar a condição em aberto.

Exemplo:

```
main()
{
  for(;;)
    printf("loop infinito");
}
```

Outra forma usual do for é o for aninhado, ou seja, um for dentro de outro.

Exemplo:

```
main()
{
```

```
  int linha,coluna;
  for(linha=1; linha<=24; linha++)
  {
    for(coluna=1; coluna<40; coluna++)
      printf("-");
      putchar('
  }
}
```

while

O comando while permite a execução de uma sequencia de comandos de forma repetitiva até que uma condição seja falsa.

Sintaxe:
```
while (expressão) {
<comandos>;
}
```

Observações:

- <comandos> pode ser vazio, simples ou bloco
- É executado sempre que a condição for verdadeira
- Não será executado se a expressão for inicialmente falsa.

Exemplo:
```
main()
{
  char ch;
  while (ch!='a') {
    ch=getchar();
  }
}
```

do...while

O comando do...while é semelhante ao comando while. A unica diferença é que a avaliação da expressão é feita ao final da execução do bloco de comandos.

Sintaxe:
```
do
{
```

```
  <comandos>;
} while (condição);
```

Observações:

- <comandos> pode ser vazio, simples ou bloco.
- Será executado ao menos uma vez, ou seja, se a expressão for inicialmente falsa.

Exemplo:

```
main()
{
  char ch;
  printf("1. inclusão");
  printf("2. alteração");
  printf("3. exclusão");
  printf(" Digite sua opção:");
  do
  {
    ch=getchar();
    switch(ch)
    {
      case '1':
        printf("escolheu inclusao");
        break;
      case '2':
        printf("escolheu alteracao");
        break;
      case '3':
        printf("escolheu exclusao");
        break;
      case '4':
        printf("sair");
    }
  } while (ch!='1' && ch!='2' && ch!='3' && ch!='4');
}
```

break

O fluxo de controle de uma estrutura de repetição pode ser alterado com o comando break. O comando break produz uma saída imediata da estrutura.

Só podemos utilizar o comando break com os comandos for, while, do...while ou switch.

Exemplo:

```
main()
{
  char ch;
  for(;;)
  {
    ch=getchar();
    if (ch=='a') break;
  }
}
```

continue

Algumas vezes torna-se necessário "saltar" uma parte do programa, para isso utilizamos o "continue".

O comando continue:

- força a próxima iteração do loop
- pula o código que estiver em seguida

Exemplo:

```
main()
{
  int x;
  for(x=0;x<100;x++)
  {
    if(x%2)
      continue;
    printf("%d");
  }
}
```

goto e labels

return

O comando return termina a execução da função atual e devolve o controle para a função chamadora.

Sintaxe: `return <expressão>;`

Onde a expressão é opcional e será o valor retornado pela função.

nulo

O comando nulo não faz nada! É representado por um ponto-e-virgula ";". Utilizamos o comando nulo quando todo o processamento necessário esta dentro da expressão de controle de repetição

Exemplo:

Funções

É uma unidade autonoma de código do programa é desenhada para cumprir uma tarefa particular. Geralmente os programas em C consistem em várias pequenas funções. A declaração do tipo da função é obrigatória no C do UNIX. Os parâmetros de recepção de valores devem ser separados por vírgulas.

Sintaxe:
```
tipo nome(parâmetros);
{ comandos}
```

Função sem Retorno

Quando uma função não retorna um valor para a função que a chamou ela é declarada como **void**.

Exemplo:
```
void inverso();
main()
{
char *vet="abcde";
inverso(vet);
}
```

```
void inverso(s)
char *s;
{
int t=0;
for(;*s;s++,t++);
s--;
for(;t--;)printf("%c",*s--);
putchar('
}
```

Função com Retorno

O tipo de retorno da função deve ser declarado.

Exemplo:

```
int elevado();
main()
{
  int b,e;
  printf("Digite a base e expoente x,y : ");
  scanf("%d,%d",&b,&e);
  printf("valor=%d
}
int elevado(base,expoente)
int base,expoente;
{
  int i;
  if (expoente<0) return;
  i=1;
  for(;expoente;expoente--)i=base*i;
  return i;
}
```

Parâmetros Formais

Quando uma função utiliza argumentos, então ela deve declarar as variáveis que aceitaram os valores dos argumentos, sendo essas variáveis os parâmetros formais.

Exemplo:
```
int pertence(string,caracter) /* pertence(char
*string,char caracter) */
char *string,caracter;
{
  while (*string)
    if (*string==caracter) return 1;
    else string++;
  return 0;
}
```

Chamada por Valor

O valor de um argumento é copiado para o parâmetro formal da função, portanto as alterações no processamento não alteram as variáveis.

Exemplo:
```
int sqr();
main()
{
  int t=10;
  printf("%d %d",sqr(t),t);
}
int sqr(int x)
{
  x=x*x;
  return x;
}
```

Chamada por Referência

Permite a alteração do valor de uma variável. Para isso é necessário a passagem do endereço do argumento para a função.

Exemplo:
```
void troca();
```

```
main()
{
  int x=10,y=20;
  troca(&x,&y);
  printf("x=%d y=%d
}
void troca(int *a,int *b)
{
  int temp;
  temp=*a;
  *a=*b;
  *b=temp;
}

int tamanho;
printf("Digite o tamanho: ");
scanf("%d",&tamanho);
linha(tamanho);
}
void linha(x)
int x;
{
int i;
for(i=0;i<=x;i++)putchar(95);
/* A variável i na função linha não é reconhecida pela
função main.*/
}
```

O Pré-processador

#include

Inclui o arquivo para compilação.

#define

Permite definir uma constante simbólica para o pré-processador.

#undef

Permite apagar a definição de uma constante simbólica definida para o processador.

#ifdef

Verifica se uma constate simbólica esta definida para o pré-processador.

#ifndef

Verifica se uma constante simbólica não esta definida para o pré-processador.

#if

#else

#elif

#endif

Permitem implementar estruturas de teste (if... else, if ... else if ..., etc)

Exemplo:
```
#if expressão1
...código...
#elif expressão2
...código alternativo1...
#else
...codigo alternativo2...
#endif
```

Exemplo:
```
#define CPU=286
...
#if CPU=386
#include "wait.h"
  wait_msec(100);
```

```
#endif
...
```

#pragma

Permite passar parâmetros não definidos no padrão ISO para o compilador.

CAPÍTULO

2

A BIBLIOTECA PADRÃO C/C++

Biblioteca de Funções

Processamento de Caracteres

isaplha

```
#include <ctype.h>
int isalpha(int c);
```

Descrição: Macro. Caso c seja uma letra ('A' a 'Z' ou 'a' a 'z'), retorna um valor diferente de zero indicando verdadeiro (true). Caso contrario, retorna zero indicando falso (false).

Exemplo:

```
#include <stdio.h>
#include <ctype.h>
#include "string.h"

main()
{
  char umaString[10];
  int i;

  strcpy(umaString,"a0c1B3ZXa4");
```

```
  for(i=0;i < 10;i++)
    if(isalpha(umaString[i]))
      printf("umaString[%d] e uma letra\n",i);
    else
      printf("umaString[%d] e um digito\n",i);

  return(0);
}
```

isdigit

```
#include <ctype.h>
int isdigit(int c);
```

Descrição: Macro. Caso c seja um digito ('0' a '9'), retorna um valor diferente de zero indicando verdadeiro (true). Caso contrario, retorna zero indicando falso (false).

Exemplo:

```
#include <stdio.h>
#include <ctype.h>
#include "string.h"

main()
{
  char umaString[10];
  int i,cont;

  strcpy(umaString,"a0c1B3ZXa4");

  for(cont=0,i=0;i < 10;i++)
    if(isdigit(umaString[i]))
      cont++;

  printf("umaString possui %d digitos",cont);
```

```
  return(0);
}
```

islower

```
#include <ctype.h>
int islower(int c);
```

Descrição: Macro. Retorna um valor diferente de zero (verdadeiro/true) indicando que c e uma letra minuscula ('a' a 'z'). Retorna zero (falso/false) caso contrario.

Exemplo:

```
#include <stdio.h>
#include <ctype.h>
#include "string.h"

main()
{
  char umaString[10];
  int i,cont;

  strcpy(umaString,"a0c1B3ZXw4");

  for(cont=0,i=0;i < 10;i++)
    if(islower(umaString[i]))
      cont++;

  printf("umaString possui %d letras
minusculas",cont);

  return(0);
}
```

isupper

```
#include <ctype.h>
int isupper(int c);
```

Descrição: Macro. Retorna um valor diferente de zero (verdadeiro/true) indicando que c e uma letra maiuscula ('A' a 'Z'). Retorna zero (falso/false) caso contrario.

Exemplo:

```
#include <stdio.h>
#include <ctype.h>
#include "string.h"

main()
{
  char umaString[10];
  int i,cont;

  strcpy(umaString,"a0c1B3ZXw4");

  for(cont=0,i=0;i < 10;i++)
    if(isupper(umaString[i]))
      cont++;

  printf("umaString possui %d letras
maiusculas",cont);

  return(0);
}
```

isspace

```
#include <ctype.h>
int isspace(int c)
```

Descrição: Macro. Retorna um valor diferente de zero (verdadeiro/true) indicando que c e um espaco, tab(\t), retorno do carro(\r), nova linha(\n), tab vertical ou salto de pagina(\f) (caracteres 09H a 0DH e 20H). Retorna zero (falso/false) caso contrario.

Exemplo:

```
#include <stdio.h>
#include <ctype.h>
#include "string.h"
```

```
main()
{
  char umaString[80];
  int i,cont;

  strcpy(umaString,"voce\tdeve\tser\nlouco\npor nao\
fgostar\n\rdesta\tlinguagem");

  for(cont=0,i=0;i < 80;i++)
    if(isspace(umaString[i]))
      cont++;

  printf("umaString possui %d caracteres
espaco",cont);

  return(0);
}
```

toascii

```
#include <ctype.h>
int toascii(int c);
```

Descrição: Macro. Retorna o valor ASCII do inteiro c zerando todos os bits do valor exceto os 7 bits menos significativos. Isto coloca o valor retornado na faixa de 0 a 127. c permanece inalterado.

Exemplo:

```
#include <stdio.h>
#include <ctype.h>

int main(void)
{
  int number, result;
  number = 511;
  result = toascii(number);
  printf("%d %d\n", number, result);
```

```
  return 0;
}
```

tolower

```
#include <ctype.h>
int tolower(int ch);
```

Descrição: Função. Caso ch possua o valor de uma letra maiuscula retorna o valor do inteiro ch convertido para o seu equivalente letra minuscula ('a' a 'z'). Valores de ch diferentes de letras maiusculas são retornados sem alteração. ch não é alterado.

Exemplo:

```
#include <string.h>
#include <stdio.h>
#include <ctype.h>

int main(void)
{
  int length, i;
  char *string = "ISTO E UMA STRING";

  length = strlen(string);
  for (i=0; i<length; i++)
    string[i] = tolower(string[i]);

  printf("%s\n",string);

  return 0;
}
```

toupper

```
#include <ctype.h>
int toupper(int ch);
```

Descrição: Função. Caso ch possua o valor de uma letra minuscula retorna o valor do inteiro ch convertido para o seu equivalente letra maiuscula ('A' a 'Z').

Valores de ch diferentes de letras minusculas sao retornados sem alteracao. ch nao e alterado.

Exemplo:

```
#include <string.h>
#include <stdio.h>
#include <ctype.h>

int main(void)
{
  int length, i;
  char *string = "isto e uma string";

  length = strlen(string);
  for (i=0; i<length; i++)
  string[i] = toupper(string[i]);

  printf("%s\n",string);

  return 0;
}
```

Processamento de Strings

strcat

```
#include <string.h>
char *strcat(char *string1,char *string2);
```

Descrição: Concatena duas strings. Não verifica tamanho.

Exemplo:

```
main()
{
  char um[20],dois[10];
  strcpy(um,"bom");
  strcpy(dois," dia");
  strcat(um,dois);
  printf("%s");
```

```
}
```

strncat

```
#include <string.h>
char *strncat(char *destino, const char *origem, long
int maxlen);
```

Descrição: strncat copia maxlen caracteres de origem para o final de destino, acrescentando em seguida um caracter de finalização '\0'. O comprimento maximo da string resultante e strlen(destino) + maxlen.

Exemplo:

```
#include <string.h>
#include <stdio.h>

int main(void)
{
  char destino[25];
  char *fonte = " Aranha come mosca";

  strcpy(destino, "Homen");
  strncat(destino, fonte, 7);
  printf("%s\n", destino);
  return 0;
}
```

strcmp

```
#include <string.h>
int strcmp(char *string1,char *string2);
```

Descrição: Compara duas strings, se forem iguais devolve 0.

Exemplo:

```
main()
{
  char s[80];
  printf("Digite a senha:");
  gets(s);
  if (strcmp(s,"laranja"))
```

```
    printf("senha inválida");
  else
  printf("senha ok!");
}
```

strncmp

```
#include <string.h>
int strncmp (const char *s1, const char *s2, long int
maxlen);
```

Descrição: Compara s1 com s2, procurando no maximo maxlen caraccteres. A comparacao tem inicio no primeiro caracter de cada string, continuando nos caracteres seguintes até que que sejam encontrados caracteres diferentes ou até que maxlen caracteres tenham sido examinados.

Retorna um valor do tipo int baseado no resultado da comparação de s1 com s2.

< 0 se s1 < s2

== 0 se s1 == s2

> 0 se s1 > s2

Exemplo:

```
#include <string.h>
#include <stdio.h>

int  main(void)

{
  char *buf1 = "aaabbb", *buf2 = "bbbccc", *buf3 =
"ccc";
  int ptr;

  ptr = strncmp(buf2,buf1,3);
  if (ptr > 0)
    printf("buffer 2 e maior do que buffer 1\n");
  else
    printf("buffer 2 e menor do que buffer 1\n");
```

```
  ptr = strncmp(buf2,buf3,3);
  if (ptr > 0)
    printf("buffer 2 e maior do que buffer 3\n");
  else
    printf("buffer 2 e menor do que buffer 3\n");

  return(0);
}
```

strcpy

```
#include <string.h>
char *strcpy(char *string1, char *string2);
```

Descrição: Copia o conteúdo de uma string.

Exemplo:

```
main(){
  char str[80];
  strcpy(str,"alo");
  puts(str);
}
```

strncpy

```
#include <string.h>
char *strncpy(char *destino, const char *origem, long
int maxlen);
```

Descrição: copia no maximo maxlen caracteres de origem para destino. destino só sera finalizado com '\0' se o comprimento de origem for menor do que maxlen.

Exemplo:

```
#include <stdio.h>
#include <string.h>

int main(void)
{
  char string[10];
  char *str1 = "abcdefghi";
```

```
  strncpy(string, str1, 3);
  string[3] = '\0';
  printf("%s\n", string);
  return 0;
}
```

strlen

```
#include <string.h>
size_t strlen(char *string);
```

Descrição: strlen calcula o comprimento da string s. Retorna o numero de caracteres em s, nao incluindo na contagem o caractere de terminação '\0'.

Exemplo:

```
#include <stdio.h>
#include <string.h>

int main(void)
{
  char *string = "CEFETES - Uned SERRA";

  printf("%d\n", strlen(string));
  return 0;
}
```

strchr

```
#include <string.h>
char *strchr(char *string, int c);
```

Descrição: procura na string s a primeira ocorrencia do caracter c. Se encontrou c em s, retorna um ponteiro para a primeira ocorrencia de c. Caso contrario retorna null. O caraccter '\0' é considerado como parte do conteudo da string s.

Exemplo:

```
#include <string.h>
#include <stdio.h>

int main(void)
```

```
{
  char string[15];
  char *ptr, c = 'r';

  strcpy(string, "Isto e uma string");
  ptr = strchr(string, c);
  if (ptr)
    printf("O caractere %c esta na posicao: %d\n", c,
ptr-string);
  else
    printf("Caracter nao foi encontrado\n");
  return 0;
}
```

strstr

```
#include <string.h>
char *strstr(char *string1, char *string2);
```

Descrição: procura na string s1 a primeira ocorrencia da (sub)string s2. Se encontrou, retorna um ponteiro para o primeiro elemento de s1 onde s2 começa (aponta para s2 em s1). Se nao encontrou (s2 não ocorre em s1), retorna null.

Exemplo:

```
#include <stdio.h>
#include <string.h>

int main(void)
{
  char *str1="CEFETES-Uned SERRA", *str2="Uned", *ptr;

  ptr = strstr(str1, str2);
  printf("A substring e: %s\n", ptr);
  return 0;
}
```

strtok

```
#include <string.h>
char *strtok(char *string1, char *string2);
```

Descrição: Função. Procura um caractere em uma string. Quando encontra retorna um ponteiro para o primeiro caractere do token, caso contrário retorna null.

Exemplo:

```
...
char string1[81]=”!primeiro token!segundo token!”;
char *ptr;
...
ptr=strtok(s1, “!”);
...
```

atof

```
#include <stdlib.h>
double atof(char *string);
```

Descrição: Converte a string numerica em s para um valor do tipo double. atof reconhece uma string como valida para conversao desde que ela corresponda ao formato abaixo

[caracteres espaco em branco] [sinal] [digitos] [.] [digitos] [[e ou E [sinal] digitos]

O primeiro caractere inválido encontrado cancela a conversao. Neste caso, retorna o valor zero.

Exemplo:

```
#include <stdlib.h>
#include <stdio.h>

int main(void)
{
  float f;
  char *str = "12345.67";

  f = atof(str);
  printf("string = %s float = %f\n", str, f);
  return 0;
```

```
}
```

atoi

```
#include <stdlib.h>
int atoi(char *string);
```

Descrição: Macro. Converte a string em s para um valor do tipo int. atoi reconhece uma string como valida para a conversão desde que corresponda ao seguinte formato

[caracteres espaco em branco] [sinal] [digitos]

O primeiro caractere invalido encontrado cancela a conversao. Neste caso, o valor retornado e zero.

Exemplo:

```
#include <stdlib.h>
#include <stdio.h>

int main(void)
{
  int n;
  char *str = "12345.67";

  n = atoi(str);
  printf("string = %s int = %d\n", str, n);
  return 0;
}
```

atol

```
#include <stdlib.h>
long atol(char *string);
```

Descrição: Macro. Converte a string em s para um valor do tipo long int. atol reconhece uma string como valida para a conversao desde que corresponda ao seguinte formato [caracteres espaco em branco] [sinal] [digitos]

O primeiro caractere invalido encontrado cancela a conversao. Neste caso, o valor retornado e zero.

Exemplo:

```
#include <stdlib.h>
#include <stdio.h>
```

```
int main(void)
{
   long l;
   char *lstr = "98765432";

   l = atol(lstr);
   printf("string = %s long int = %ld\n", lstr, l);
   return(0);
}
```

Alocação e Manipulação de Memória

memchr

```
#include <string.h>
void *memchr(void *endereco, int c, size_t n);
```

Descrição: Procura pela primeira ocorrencia do caractere c, iniciando da posição de memória indicado pelo ponteiro endereço. A função retorna após encontrar o caractere c ou o valor de caracteres procurados atigir o valor n). A função retorna o endereço do caractere encontrado.

Exemplo:
```
char mensagem="Esta é a cadeia de carateres que vai ser pesquisada";
...
char *ptr;
ptr=memchr(mensagem, 'm', sizeof(mensagem);
```

memcmp

```
#include <string.h>
int memcmp(void *endereco1, void *endereco2, size_t n);
```

Descrição: Compara o conteúdo da memoria apontado pelo endereco1 com o conteúdo apontado por endereco2 (um byte de cada vez). A função retorna quando encontra dois valores diferentes ou apos comparar n valores.

retorna < 0 se *endereco1 < *endereco2

retorna = 0 se *endereco1 = *endereco2

retorna > 0 se *endereco1 > *endereco2

Exemplo:

```
...
gets(buf1);
gets(buf2);
i = memcmp(buf1, buf2, sizeof(buf1));
...
```

memcpy

```
#include <string.h>
int memcpy(void *end_destino, void *end_origem, size_t
n);
```

Descrição: Copia n bytes do endereço de memória apontado por `end_origem` para o endereço apontado por `end_destino`.

Exemplo:

```
...
double x[100], y[100];
...
memcpy(y, x, sizeof(x));
...
```

memset

```
#include <string.h>
void *memset(void *endereco, int c, size_t n);
```

Descrição: Preenche n bytes da memória com o valor `c` a partir do endereço apontado por `endereco`.

Exemplo:

```
...
int i, array[1000];
...
memset(array, 0, sizeof(array));
...
```

malloc

```
#include <stdlib.h>
void *malloc(size_t tamanho);
```

Descrição: Aloca na memória um numero de bytes igual a `tamanho`. A função retorna o endereço de memória do primeiro byte alocado ou então um valor `null` caso não seja possivel fazer a alocação.

Exemplo:
```
int *nomes[1000];
char buffer[80];
...
for(i=0; i<1000; i++)
{
  printf("Entre com um nome")
  buffer[0]='\0';
  gets(buffer);
  tam=sizeof(buffer);
  nomes[i] = malloc(tam);
  ...
}
```

calloc

```
#include <stdlib.h>
void *calloc(size_t numero, size_t tamanho);
```

Descrição: Aloca espaço na memória para um array com um número de elementos igual a `numero` tendo cada elemento o `tamanho` em bytes. A função retorna o endereço de memória do primeiro byte alocado ou então um valor `null` caso não seja possivel fazer a alocação.

Exemplo:
```
...
#define NUMERO 5
struct _nome {
  char primeiro[15];
  char meio[15];
  char sobrenome[15];
}
...
main()
{
  int i;
```

```
  struct _nome (*nome)[ ]; // ponteiro para um array
                           // de estruturas
  nome=calloc(NUMERO, sizeof(struct _nome));
  ...
}
```

free

```
#include <stdlib.h>
void free(void *ponteiro);
```

Descrição: Desaloca a área de memória apontada por `ponteiro`.

Exemplo:

```
...
free(dados);
...
```

realloc

```
#include <stdlib.h>
void *realloc(void *ponteiro, size_t tamanho);
```

Descrição: Realoca um bloco de memória alocado previamente pelas funções malloc, calloc ou realloc. O novo bloco de memoria terá `tamanho` bytes. O conteúdo do bloco permanece intacto apos a realocação.

Exemplo:

```
...
#define TAMANHO 10
main( ) {
  int i, c, tamanhoLinha = TAMANHO;
  char (*buffer)[ ]; // ponteiro para um array de char
  if (buffer = realloc(NULL, tamanhoLinha)) != NULL) {
    printf("Este programa le caracteres até que return
seja pressionado");
    for(i=0; (c=getchar()) != '\n'; i++) {
      if (i == tamanhoLinha-1) {
        tamanhoLinha += TAMANHO;
        // aumenta o tamanho de buffer
        buffer = realloc(buffer, tamanhoLinha);
```

```
        if (buffer == NULL) {
          printf("Memória insuficiente")
          exit(1);
        }
      }
      (*buffer)[i]=c; // coloca o caractere lido no
buffer
    }
    (*buffer)[i]='\0'; // coloca nulo para terminar a
cadeia
  ...
}
```

Entrada e Saída

FILE

Descrição: O tipo FILE é utilizado para representar as informações de controle para um arquivo.

Exemplo: **`FILE *ptrArquivo;`**

EOF

Descrição: É o valor padrão utilizado para o final de arquivo.

Exemplo:

fopen

```
#include <stdio.h>
FILE *fopen(char *nome_arquivo, char *acesso)
```

Descrição: Abre um arquivo com o nome especificado em `nome_arquivo` e com o tipo de acesso indicado por `acesso`.

O acesso é definido por:

r leitura somente

w escrita

a permite escrever no final do arquivo.

rb leitura de arquivo binário

wb escrita de arquivo binário

ab escreve no final de um arquivo binário

r+ abre arquivo de texto para atualização (leitura/escrita)

w+ abre/cria arquivo de texto para atualização (leitura/escrita)

a+ igual a w+, porém apos o final do arquivo

r+b abre arquivo binário para atualização (leitura/escrita)

w+b abre/cria arquivo binário para atualização (leitura/escrita)

a+b igual a w+b, porém no final do arquivo

A função retorna um ponteiro associado ao arquivo.

Exemplo:

```
...
struct DADOS {
  char string[22];
  int numero;
};
main( ) {
  int i;
  FILE *fp;
  struct DADOS record = {"Registro numero", 0};
  fp = fopen("arquivo.dados", "w+b");
  if (fp != NULL) {
    ...
  }
...
}
```

fclose

```
#include <stdio.h>
int fclose(FILE *ponteiro_arquivo);
```

Descrição: Fecha o arquivo associado ao ponteiro.

Exemplo:

```
...
  FILE *fp;
```

```
  ...
  if (fclose(fp) != 0) {
    printf("Erro ao fechar o arquivo!);
  }
...
```

stdin

Descrição: Macro. É um ponteiro para um objeto do tipo FILE, e aponta para a entrada padrão do sistema.

Exemplo:

stdout

Descrição: Macro. É um ponteiro para um objeto do tipo FILE, e aponta para a saida padrão do sistema.

Exemplo:

stderr

Descrição: Macro. É um ponteiro para um objeto do tipo FILE, e aponta para a saida padrão de erros do sistema.

Exemplo:

fseek

```
#include <stdio.h>
int fseek(FILE *ponteiro_arquivo, long offset, int origem)
```

Descrição: Aponta o indicador de posição do arquivo para o valor especificado em offset e origem. Dependendo do tipo do arquivo o valor de origem pode ter significado diferente:

- Arquivos Binários:
  - SEEK_SET indica início do arquivo
  - SEEK_CUR indica posição atual
  - SEEK_END indica final do arquivo
- Em arquivos texto:
  - SEEK_SET pode ser zero

- ou o valor retornado pela função ftell( )

Exemplo:

```
// Le cada quinto caractere do arquivo
...
FILE *ptrArq;
...
long int offset = 5;
...
ptrArq = fopen("teste.txt", "rb");
fseek(ptrArq, 0, SEEK_END);
fgetpos(ptrArq, &end_p);
rewind(ptrArq);
fgetpos(ptrArq, &curr_p);
while(curr_p < end_p) {
  c = getc(ptrArq);
  putchar(c);
  fseek(ptrArq, offset, SEEK_CUR);
  fgetpos(ptrArq, &curr_p)
}
fclose(ptrArq);
...
}
```

fgetc

```
#include <stdio.h>
int fgetc(FILE *ponteiro_arquivo);
```

Descrição: Função. Retorna o proximo caractere apontado pelo ponteiro_arquivo.

Exemplo:

```
...
char in;
...
while ( (in=fgetc(stdin) ) != EOF ) {
  printf("%c", in);
}
```

```
...
```

getc

```
#include <stdio.h>
int getc(FILE *ponteiro_arquivo);
```

Descrição: Macro. Implementação em macro equivalente a fgetc( ).

Exemplo:

```
...
while ( (in=getc(stdin) ) != EOF )
...
```

getchar

```
#include <stdio.h>
int getchar(void);
```

Descrição: Função. Retorna um caractere lido da entrada padrão (stdin).

Exemplo:

```
main()
{
char ch;
ch=getchar();
printf("%c
}
```

fgets

```
#include <stdio.h>
char *fgets(char *buffer, int n, FILE *ptr_arquivo);
```

Descrição: Função. Lê uma cadeia de caracteres do arquivo apontado por *ptr_arquivo e armazena na cadeia apontada por *buffer. fgets le n-1 caracteres

Exemplo:

```
...
  FILE *ptrArq;
  char str[256];
  ptrArq = fopen("teste.txt", "r");
  while( !feof( ptrArq ) ) {
    fgets(str, 256, ptrArq);
    printf("%s", str);
```

```
  }
  fclose(ptrArq);
...
```

gets

```
#include <stdio.h>
char *gets(char *buffer);
```

Descrição: Função. É utilizada para leitura de uma string através do dispositivo padrão, até que o ENTER seja pressionado.

Exemplo:

```
main()
{
char str[80];
gets(str);
printf("%s",str);
}
```

fscanf

```
#include <stdio.h>
int fscanf(FILE *ptr_arquivo, char *formato1, ...)
```

Descrição: Função. Le dados do arquivo apontado por ptr_arquivo de acordo com o formato especificado

Exemplo:

```
...
fscanf( ptrArq, "%d %s %s %[1-9] %lf %Lf",
        &num_emp, primeiro, sobrenome, &classficacao,
        &horas, &valor_hora);
...
```

scanf

```
#include <stdio.h>
int scanf(char *formato, ...);
```

Descrição: Função. Identica a função fscanf( ) porém com o arquivo de entrada apontado para a entrada padrão (stdin).

Exemplo:

```
...
int num;
```

```
printf("Digite um número: ");
scanf("%d",&num);
printf("O numero digitado foi: %d", num);
...
```

fputc

```
#include <stdio.h>
int fputc(int c, FILE *ptr_arquivo);
```

Descrição: Função. Escreve um caractere no arquivo apontado por ptr_arquivo. O inteiro é convertido para `unsigned char` antes da escrita.

Exemplo:

```
// programa para copia arquivos
...
FILE *ptr_origem;
FILE *ptr_destino;
int c;
ptr_origem=fopen("origem.txt", "r");
ptr_destino=fopen("destino.txt", "w");
while ( (c=fgetc(ptr_origem) ) != EOF ) {
  fputc(c, ptr_destino);
}
...
```

putc

```
#include <stdio.h>
int putc(int c, FILE *ponteiro_arquivo);
```

Descrição: Macro. Implementação em macro da função fputc( ).

Exemplo:

```
...
putc(char_buffer, stdout);
...
```

putchar

```
#include <stdlib.h>
int putchar(int c);
```

Descrição: Escreve na tela o argumento de seu caractere na posição corrente.

Exemplo:
```
main()
{
char ch;
printf("digite uma letra minúscula : ");
ch=getchar();
putchar(toupper(ch));
}
```

fputs

```
#include <stdio.h>
int fputs(char *string, FILE *ptr_arquivo);
```

Descrição: Função. Escreve uma cadeia de caracteres apontada por *string no arquivo apontado por *ptr_arquivo.

Exemplo:
```
...
while(gets(str) != NULL) {
  fputs(str, ptrArq);
  fputs("\n", ptrArq);
}
...
```

puts

```
#include <stdio.h>
puts(char *string);
```

Descrição: Escreve o seu argumento no dispositivo padrão de saída (vídeo).

Exemplo:
```
main()
{
  puts("mensagem");
}
```

fprintf

```
#include <stdio.h>
int fprintf(FILE *ptr_arquivo, char *formato, ...);
```

Descrição: Função. Escreve uma saída formatada no arquivo apontado por *ptr_arquivo.

Exemplo:

```
...
while(scanf("%s %s %d",nome,sobrenome,&idade)!=EOF) {
  fprintf(ptrArq, "%s %s %d", nome, sobrenome, idade)
}
...
```

printf

```
#include <stdio.h>
int printf(char *formato, ...);
```

Descrição: Função. Idêntica a `fprintf` com o arquivo de saída para `stdout`.

Exemplo:

fread

```
#include <stdio.h>
size_t fread(void *buffer, size_t tamanho, size_t
numero, FILE *ponteiro_arquivo);
```

Descrição: Função. Lê dados do arquivo associado ao *ponteiro_arquivo e coloca no aray apontado por *buffer. O tamanho dos dados é fornecido no parâmetro `tamanho`. O número de membros a ser escrito é fornecido no parâmetro `numero`.

Exemplo:

```
...
do {
  loop=fread(vetor, tamanho, n_memb, ptrArq);
  for(i=0; i<=loop-1; i++)
    putchar(vetor[i]);
} while (loop==n_memb);
...
```

fwrite

```
#include <stdio.h>
size_t fwrite(void *buffer, size_t tamanho, size_t
numero, FILE *ponteiro_arquivo);
```

Descrição: Função. Escreve os ados apontaos por *buffer no arquivo associado a *ponteiro_arquivo. O tamanho dos dados é fornecido no parâmetro tamanho. O número de membros a ser escrito é fornecido no parâmetro numero.

Exemplo:
```
double d[10];
...
file_ptr=fopen("dados.dat", "wb");
if(fwrite(d, sizeof(d), 1, file_ptr)==1)
  printf("OK");
else
  printf("Falha");
...
```

feof

```
#include <stdio.h>
int feof(FILE *ponteiro_arquivo);
```

Descrição: Função. Testa a condição de final de arquivo.

Exemplo:
```
FILE *ptrArq;
char c;
...
while(!feof(ptrArq)) {
  c=fgetc(ptrArq);
  putchar(c);
}
...
```

ferror

```
#include <stdio.h>
int ferror(FILE *ponteiro_arquivo);
```

Descrição: Função. Teste a condição de erro do arquivo apontado por *ponteiro_arquivo.

Exemplo:
```
FILE *ptrArq;
char c;
...
```

```
while(!ferror(ptrArq) && !feof(ptrArq)) {
  c=fgetc(ptrArq);
  putchar(c);
}
...
```

remove

```
#include <stdio.h>
int remove(char *nome_arquivo);
```

Descrição: Função. Deleta um arquivo.

Exemplo:
```
i=remove("win.com");
if(i==0)
  printf("Sucesso: arquivo removido");
else
  printf("Falha: arquivo não removido");
```

rename

```
#include <stdio.h>
int rename(char *nome_antigo, char *nome_novo);
```

Descrição: Função. Permite renomear arquivos.

Exemplo:
```
i=rename("c:velho.txt","c:novo.txt");
if(i==0)
  printf("Sucesso: arquivo renomeado");
else
  printf("Falha: arquivo não renomeado");
```

Matemáticas

abs

```
#include <stdlib.h>
int abs(int n);
```

Descrição: Calcula o valor absoluto de x.

Exemplo:
```
c=abs(-10);
```

fabs

```
#include <math.h>
double fabs(double x);
```

Descrição: Função. Calcula o valor absoluto de um número em ponto flutuante.

Exemplo: `r=fabs(x);`

ceil

```
#include <math.h>
double ceil(double x);
```

Descrição: Função. Retorna o menor inteiro que é maior que x.

Exemplo: `r=ceil(x);`

floor

```
#include <math.h>
double floor(double x);
```

Descrição: Função. Retorna um número de precisão dupla que representa o maior inteiro que é menor ou igual a x.

Exemplo: `r=floor(x);`

fmod

```
#include <math.h>
double fmod(double x, double y);
```

Descrição: Função. Retorna o valor em ponto-flutuante do resto da divisão de x/y.

Exemplo: `r=fmod(x,y);`

exp

```
#include <math.h>
double exp(double x);
```

Descrição: Calcula o valor de e^x.

Exemplo: `r=exp(x);`

log

```
#include <math.h>
double log(double x)
```

Descrição: Calcula o valor do logaritmo natural (neperiano) de x.

Exemplo:
```
r=log(x);
```

log10

```
#include <math.h>
double log10(double x);
```

Descrição: Calcula o valor do logaritmo decimal de x.

Exemplo:
```
r=log10(x);
```

pow

```
#include <math.h>
double pow(double x, double y);
```

Descrição: Calcula o valor de x^y, isto é x elevado a y.

Exemplo:
```
r=pow(2,3);
```

pow10

```
#include <math.h>
double pow10(int x);
```

Descrição: Calcula o valor de 10^x, isto é 10 elevado a x.

Exemplo:
```
r=pow10(2);
```

sqrt

```
#include <math.h>
double sqrt(double x);
```

Descrição: Função. Calcula a raiz quadrada de x (x deve ser positivo).

Exemplo:
```
r=sqrt(4);
```

rand

```
#include <stdlib.h>
int rand(void);
```

Descrição: Função. Gera um número pseudo-randômico. Os números gerados estão na faixa entre 0 e RAND_MAX.

Exemplo:
```
i=rand();
```

srand

```
#include <stdlib.h>
void srand(unsigned semente);
```

Descrição: Função. Gera a semente para o gerador de números pseudo-randômicos. Para uma mesma semente será gerada sempre a mesma sequencia.

Exemplo:
```
...
srand(10);
...
```

cos

```
#include <math.h>
double cos(double x);
```

Descrição: Função. Retorna o cosseno do angulo x (em radianos).

Exemplo:
```
r=cos(x);
```

sin

```
#include <math.h>
double sin(double x);
```

Descrição: Função. Retorna o seno do angulo x (em radianos).

Exemplo:
```
r=sin(x);
```

tan

```
#include <math.h>
double tan(double x);
```

Descrição: Função. Retorna o valor da tangente de x (em radianos).

Exemplo:
```
r=tan(x);
```

acos

```
#include <math.h>
double acos(double x);
```

Descrição: Função. Retorna o arcosseno de x (em radianos). O valor de x deve entar entre -1. e 1.

Exemplo:
```
r=acos(x);
```

asin

```
#include <math.h>
double asin(double x);
```

Descrição: Função. Retorna o arcoseno de x (em radianos). O valor de x deve estar entre -1. e 1.

Exemplo: `r=asin(x);`

atan

```
#include <math.h>
double atan(double x);
```

Descrição: Função. Calcula o arcotangente de x.

Exemplo: r=**atan(x)**;

atan2

```
#include <math.h>
double atan2(double x, double y);
```

Descrição: Função. Calcula o arcotangente de $\frac{x}{y}$.

Exemplo: `r=atan2(x,y);`

cosh

```
#include <math.h>
double cosh(double x);
```

Descrição: Função. Calcula o seno hiperbolico de x.

Exemplo: `r=cosh(x);`

sinh

```
#include <math.h>
double sinh(double x);
```

Descrição: Função. Calcula o seno hiperbolico de x.

Exemplo: `r=sinh(x);`

tanh

```
#include <math.h>
double tanh(double x);
```

Descrição: Função. Calcula a tangente hiperbólica de x (em radianos).

Exemplo:
```
r=tanh(x);
```

Data e Hora

clock

```
#include <time.h>
clock_t clock(void);
```

Descrição: Função. Calcula o tempo de execução do programa em "clock ticks". Para obter o tempo em segundos devemos dividir o valor etornado po CLK_TCK.

Exemplo:
```
clock_t t1, t1;
...
t1=clock();
...
t2=clock();
tempo=((t2-t1)/CLK_TCK);
printf("O tempo de execução: %d\n", tempo);
...
```

time

```
#include <stdio.h>
time_t time(time_t *timer);
```

Descrição: Função. Retora o valor da data atual.

Exemplo:
```
time_t t;
...
t=time();
```

time_t

Descrição: Estrutura. Tipo de dado que armazena datas.

Exemplo:

asctime

```
#include <time.h>
```

```
char *asctime(struct tm *ponteiro_time);
```

Descrição: Função. Converte a hora em uma cadeia de caracteres.

Exemplo:

```
time_t t;
struct tm *hora_local;
time(&t);
hota_local=localtime(&t);
printf("Hora local=%s", asctime(hora_local));
```

ctime

```
#include <time.h>
char *ctime(time_t *timer);
```

Descrição: Função. Converte a hora fornecida por *timer para uma cadeia de caracteres.

Exemplo:

```
time_t t;
...
time(&t);
printf("Hora Atual=%s", ctime(&t));
...
```

localtime

```
#include <time.h>
struct tm *localtime(const time_t *timer);
```

Descrição: Função. Converte a data que é apontada por `*timer` em data local.

Exemplo:

```
time_t t;
struct tm *hora_local;
time(&t);
hora_local = localtime(&t);
```

difftime

```
#include <time.h>
double difftime(time_t time1, time_t time2);
```

Descrição: Função. Calcula a diferença em segundos entre as horas time1 e time2.

Exemplo:

Controle

assert

```
#include <assert.h>
void assert(int expressão);
```

Descrição: Macro. É uma ferramenta de diagnóstico. A expressão será avaliada, se for falsa (isto é, 0) será emitida uma mensagem de erro. O assert é muito útil na depuração de programas.

Exemplo:

```
...
assert(a==3);
...
```

system

```
#include <stdlib.h>
int system(char *string_comando);
```

Descrição: Função. Chama um comando do sistema operacional.

Exemplo:

```
if (system(NULL) != 0)
  system("DIR");
```

exit

```
#include <stdlib.h>
void exit(int status);
```

Descrição: Função. Termina um programa normalmente.

Exemplo:

abort

```
#include <stdlib.h>
void abort(void);
```

Descrição: Função. Causa uma interrupção anormal do programa e aciona o sinal SIGABRT.

Exemplo:

```
...
if (erro_catastrofico) abort();
...
```

atexit

```
#include <stdlib.h>
int atexit(void (*ponteiro_função)(void));
```

Descrição: Função. Registra uma função a ser chamada quando o programa terminar normalmente.

Exemplo:

```
...
void termina1(void);
void termina2(void);
main() {
  atexit(termina1);
  atexit(termina2);
  printf("Programa principal\n");
}
void termina1(void)
{
  printf("Processando a função termina1\n");
}
void termina2(void)
{
  printf("processando a função termina2\n");
}
```

signal

```
#include <signal.h>
void *signal(int sinal, void (*sighandler(int)))(int);
```

Descrição: Função. Associa uma rotina de tratamento de sinais com um sinal.

Exemplo: Veja abaixo.

raise

```
#include <signal.h>
int raise(int signal);
```

Descrição: Função. Envia um sinal (signal) para o programa

Exemplo:

```
// exemplo de tratamento de sinal
```

```
...
void funcaox(int sinal);
...
main() {
  int i;
  signal(SIGINT, funcaox);
  signal(SIGABRT, funcaox);
  printf("Pressione CNTRL-BREAK durante o loop.\n");
  for(i=1; i<=200; i++)
    printf("Iteração numero %d.\n", i);
  printf("\nAgora o sinal será RAISE.\n");
  raise(SIGABRT);
}
void funcaox(int sinal)
{
  char ch;
  printf("Tratando o sinal\n");
  if (sinal == SIGINT)
    printf("Tratando a interrupção do usuário\n");
  else
    printf("Tratando um sinal de abort\n");
  printf("Pressione Return para continuar\n");
  ch=getchar()
  signal(SIFGINT, funcaox);
  signal(SIGABRT, funcaox);
  return;
}
```

Diversos

main

```
#include < ...>
```

```
int main() { ... }
int main(int argc, char *argv[]) { ... }
int main(int argc, char *argv[], char *envp) { ... }
```

Descrição: É a função que é automaticamente chamada quando o programa é iniciado.

Exemplo:

getenv

```
#include <stdlib.h>
char *getenv(char *nome);
```

Descrição: Função. Busca no ambiente do sistema operacional uma cadeia de caracteres que corresponda com a cadeia fornecida

Exemplo:

```
...
// procura pelo valor da variavel de ambiente PATH
// e imprime
printf("%s\n", getenv("PATH"));
...
```

bsearch

```
#include <stdlib.h>
void *bsearch(void *chave, void *base, size_t numero,
size_t tamanho, int (*compare)(void *, void *));
```

Descrição: Função. Realiza uma busca binária de um objeto em um array.

Exemplo:

```
...
typedef struct {
  char nome[20];
  char numero[10];
} LISTA_TELEFONE;
LISTA_TELEFONE amigas[] = {"Alessandra", "1111-2222",
                           "Edna", "3333-4444",
                           "Isaura", "5555-6666",
                           "Marize", "7777-8888",
                           "Solimara", "9999-0000"}
main() {
```

```
  LISTA_TELEFONE *busca;
  LISTA_TELEFONE *base;
  size_t namigas;
  size_t tamanho;
  char *chave;
  base=amigas;
  namigas=sizeof(amigas)/sizeof(amigas[0]);
  tamanho=sizeof(LISTA_TELEFONE);
  chave="Isaura";
  busca=bsearch(chave,base,namigas,tamanho,strcmp);
  if(busca == NULL)
    printf("Não encontrado");
  else
    printf(" %s %s\n", busca->nome, busca->numero);
}
```

qsort

```
#include <stdlib.h>
void qsort(void *base, size_t numero, size_t tamanho,
int (*compare)(void *, void *));
```

Descrição: Função. Classifica um array de objetos. O array é classificado de acordo com a função de comparação fornecida

Exemplo:

```
...
char lista[5][10]={"Isaura", "Alessandra",
                   "Marize", "Solimara", "Edna");
qsort(lista, 5, sizeof(lista[0]), strcmp);
...
```