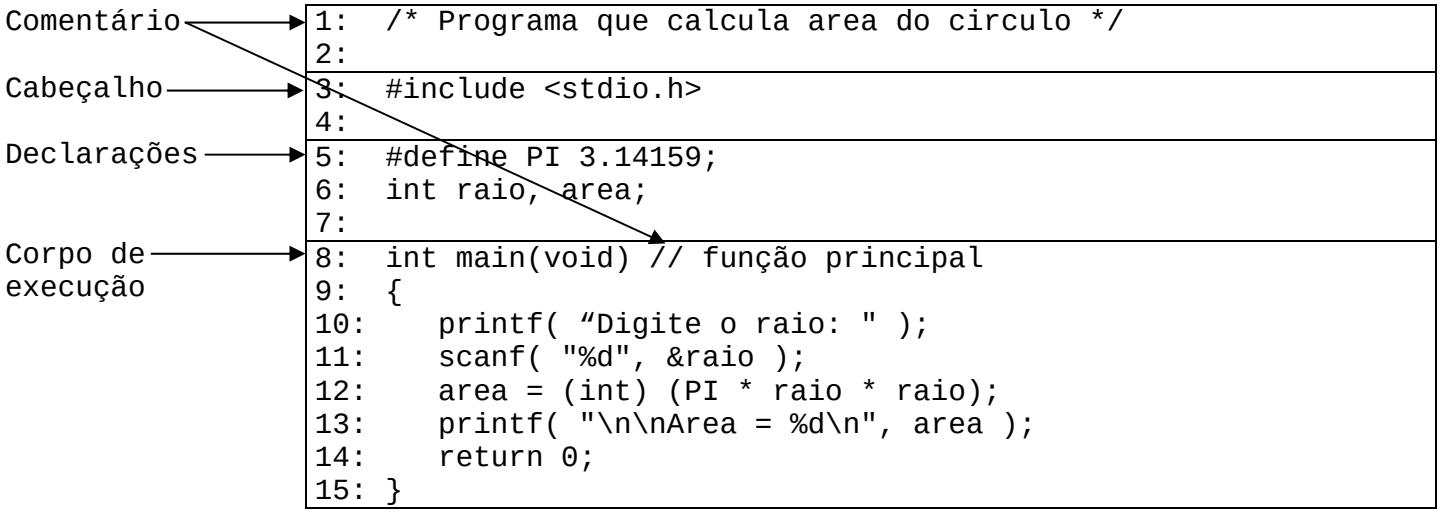
Características da Linguagem C

- Programação estruturada
  - Bloco simples
  - Compartimentalização (secciona e esconde do resto do programa instruções necessárias a uma tarefa)
- Declaração de variáveis
  - Para o compilador conhecer espaço de memória gasto
- Funções – blocos de programas
  - Permite esconder parte do código e variáveis (encapsulamento)
  - Saber o que as rotinas fazem; não como fazem.
- Testes de condição
- Laços
- Sensível ao caso (variavel ≠ Variavel)

Formato de um programa




```
1:  /* Programa que calcula area do circulo */
2:
3:  #include <stdio.h>
4:
5:  #define PI 3.14159;
6:  int raio, area;
7:
8:  int main(void) // função principal
9:  {
10:    printf( “Digite o raio: " );
11:    scanf( "%d", &raio );
12:    area = (int) (PI * raio * raio);
13:    printf( "\n\nArea = %d\n", area );
14:    return 0;
15: }
```

Elementos Básicos

Identificadores

- Usados para dar nomes as constantes, variáveis, funções e objetos
- Todo identificador deve iniciar por uma letra (a..z, A..Z) ou sublinhado
- Após o primeiro caracter, pode ser utilizados: letras, sublinhados ou dígitos
- Não pode conter símbolos especiais, ser palavra reservada nem nome de funções de bibliotecas
- Máximo de 32 caracteres

Tipos de dados básicos

Tipo	Nº de bytes	Escala
char	1	-128 a 127
int	2	-32768 a 32767
float	4	-3.4E-38 a 3.4E+38 (+/-)
double	8	-1.7E-308 a 1.7E+308 (+/-)
void	0	Sem valor

Modificadores

Tipo	Nº de bytes	Escala
unsigned char	1	0 a 255
unsigned int	2	0 a 65535
short int	2	-32768 a 32677
unsigned short int	2	0 a 65535
long int (long)	4	-2147483648 a 2147483647

Variáveis

- Valores que podem ser alterados pelo programa
- Formato: `<tipo da variável> <lista de variáveis>;`
  - `<tipo da variável>` é um tipo de dado válido em C
  - `<lista de variáveis>` são identificadores separados por vírgula
- Exemplos:
  - `int valor;`
  - `unsigned char letra, caracter;`

Constantes

- Valores fixos que não podem ser alterados pelo programa
- Formatos:
  - `#define <nome da constante> <valor>;`
  - `const <tipo da constante> <nome da constante> = <valor>;`
- Exemplos:
  - declaração global

```
#define PI 3.14;
void main(void) {
   // instrucoes do programa
}
```

  - declaração local (somente usando `const`)

```
void teste(void) {
   const float PI = 3.14;

   // instrucoes da funcao
}
```

Instruções

- Expressão comumente seguida de ponto e vírgula
- Atribuição
  - Operando do lado esquerdo é sempre uma variável
  - Operando do lado direito é de um tipo de dado compatível com o da variável

```
disciplina = "MC102";
alunos = 51;
```

- Chamada de função

```
soma(x, y);
```

- Teste de desvio.

```
if (x < 10) {
   // outras instrucoes
}
```

- Teste de repetição.

```
for (x = 0; x < 10; x++) {
   // outras instrucoes
}
```

Operadores

Operadores Aritméticos

- Definidos para ambos os tipos inteiros e não inteiros (exceto o resto "%" que não é definido p/ não inteiros)
- Erros de estouro nem sempre detectados. Cabe ao programador dimensionar corretamente suas variáveis.

Adição	+
Subtração	-
Multiplicação	*
Divisão	/
Resto da divisão	%

Operadores Relacionais

- Resultam um inteiro representando um valor lógico (1 = verdadeiro; 0 = falso)

Menor que	<
Maior que	>
Menor ou igual que	<=
Maior ou igual que	>=
Igualdade	==
Desigualdade	!=

Operadores Lógicos

- Podem receber qualquer valor de operando, sendo valores diferentes de zero interpretados como verdadeiro e iguais a zero como falso

E (conjunção)	&&
Ou (disjunção	||
Não (negação)	!

Operadores de Endereço

&	Retorna o endereço da variável
*	Retorna o conteúdo do endereço armazenado em uma variável do tipo ponteiro

Operadores de Atribuição

- O operador de atribuição em C é o sinal de "=" e pode ser usado em expressões com outros operadores.

Operadores de Atribuição Combinados

`a += b;`	`a = a + b;`
`a -= b;`	`a = a – b;`
`a *= b;`	`a = a * b;`
`a /= b;`	`a = a / b;`

Operadores Pós-fixados e Pré-fixados

++variavel	Incrementa a variável antes de usar o seu valor
variavel++	Incrementa a variável depois de usar o seu valor
--variavel	Decrementa a variável antes de usar o seu valor
variavel--	Decrementa a variável depois de usar o seu valor

Operadores Bit-a-Bit

&	E (AND)
|	Ou (OR)
^	Ou exclusivo (XOR)
~	Complemento de 1
>>	Deslocamento para a direita
<<	Deslocamento para a esquerda

Operador cast

- Força uma expressão a ser de um determinado tipo.
- Sintaxe: `(tipo) expressão;`

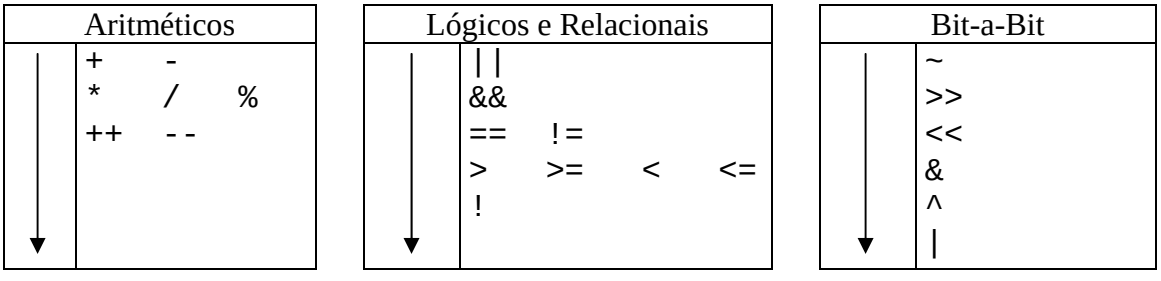
```
int i = 1;
printf("%d/3 é igual a %f", i, (float) i/3 );
```

Operador sizeof

- Retorna o tamanho em bytes da variável (tipo do operando).

```
int i;
printf("O tamanho do tipo int é %d", sizeof(i) );
```

Prioridade sobre operadores



Exercício: Qual o resultado das seguintes sentenças?

a)
```
2 * 10 % 3 – (-8)
20 % 3 – (-8)
2 – (-8)
2 + 8
10
```

b)
```
21 / (2 * abs(-8) ) * 4
21 / (2 * 8) * 4
21 / 16 * 4
1 * 4
4
```

c)
```
3 + 3 * sqrt(3 + 6) / (2 + (33 % 4))
3 + 3 * 3 / (2 + 1)
3 + 3 * 3 / 3
3 + 9 / 3
3 + 3
6
```

d)
```
!( pow(2,3) < pow(4,2) || abs(15/-2) ) < 10
!( 8 < 16 || abs(-7) ) < 10
!( 8 < 16 || 7 ) < 10
!( V || 7 ) < 10
!( V ) < 10
F < 10
V
```

e)
```
abs(17/2) != 4 + 2 && 2 + (3*5/2)%5 > 0
8 != 6 && 2 + (15/2)%5 > 0
8 != 6 && 2 + 7%5 > 0
8 != 6 && 2 + 2 > 0
8 != 6 && 4 > 0
8 != 6 && V
V && V
V
```

Funções Básicas

printf()

- Biblioteca: `stdio.h`
- Imprime caracteres numéricos e alfanuméricos no dispositivo padrão (monitor)
- Sintaxe: `printf("expressão de controle", argumentos);`
- A expressão de controle pode conter caracteres que serão exibidos na tela e os códigos de formatação que indicam o formato em que os argumentos devem ser impressos.
- Cada argumento deve ser separado por vírgula.

\n	Nova linha
\t	Tabulação horizontal
\v	Tabulação vertical
\b	Retrocesso
\”	Aspas dupla
\’	Aspas simples
\f	Alimentação de formulário
\a	Sinal sonoro
\\	Barra
\0	Nulo

%c	Caractere simples
%d	Decimal (int)
%ld	Decimal longo (long int)
%e	Notação científica
%f	Ponto flutuante (float)
%lf	Ponto flutuante longo (double)
%o	Octal
%s	Cadeia de caracteres
%u	Decimal sem sinal
%x	Hexadecimal

- Tamanho usado para os campos na impressão

```
printf("\n%7s", "Rodrigo"); // 7 caracteres: Rodrigo
printf("\n%7d", 1979);      // 7 caracteres:    1979
```

- Arredondamento

```
printf("\n%5.3f”, 3.1415); // 5 caracteres e 3 casas decimais: 3.142
printf("\n%6.2f”, 3.1415); // 6 caracteres e 2 casas decimais:  3.14
printf("\n%4.1f”, 3.1415); // 4 caracteres e 1 casa  decimal :  3.1
```

- Complementando com zeros à esquerda

```
printf("\n%04d", 21);    // 4 caracteres complementando c/ 0s: 0021
printf("\n%04.0f", 3.1); // 4 caracteres e 0 casas decimais  : 0003
```

- Imprimindo caracteres
  - Imprimir o caracter de código 3393 em ASCII é o mesmo que 3393 mod 256 = 65, ou seja, o “A”

```
printf("%d %c %x %o \n", ‘A’, ‘A’, ‘A’, ‘A’);  // 65 A 41 101
printf("%c %c %c %c \n", ‘A’, 65, 0x41, 0101); // A A A A
```

scanf()

- Biblioteca: `stdio.h`
- Lê caracteres numéricos e alfanuméricos do dispositivo padrão (teclado)
- Sintaxe: `scanf(“expressão de controle”, argumentos);`
- A lista de argumentos deve consistir nos endereços das variáveis (operador “&”)
- Cada argumento deve ser separado por vírgula.

```
int num1, num2;
printf("Digite dois números separados por um espaço: ");
scanf("%d %d", &num1, &num2);
printf("\nO inteiro %d está na posição de memória %u", num1, &num1);
printf("\nO inteiro %d está na posição de memória %u", num2, &num2);
```

getchar()

- Biblioteca: `stdio.h`
- Lê caracteres numéricos e alfanuméricos do dispositivo padrão (teclado) até que ENTER seja pressionado.

```
char ch;
ch = getchar();
printf("%c\n", ch);
```

putchar()

- Biblioteca: `stdio.h`
- Escreve na tela o argumento de seu caractere na posição corrente.

```
char ch;
printf("\nDigite uma letra minuscula: ");
ch = getchar();
putchar( toupper(ch) );  // função toupper requer biblioteca "ctype.h"
putchar('\n');
```

Exercício: Fazer um programa em linguagem C para cada um dos seguintes problemas:

a) *Trocar valores* – Receber dois números inteiros do usuário via teclado e os alocar em duas regiões distintas da memória RAM (duas variáveis do tipo `int`). Depois trocar os dois valores alocados de região, ou seja, o valor alocado na primeira região vai para a segunda e o da segunda vai para a primeira. Em seguida, imprimir no monitor o conteúdo das variáveis (agora com os valores trocados).

```
#include <stdio.h>

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

	scanf("%d %d", &a, &b);
	temp = a;
	a = b;
	b = temp;
	printf("%d %d", a, b);

	return 0;
}
```

b) *Progressão Aritm*ética – Receber um número inteiro *n* e calcular a soma de todos os números naturais até *n*: $\sum_{x=1}^{n} x$

c) *Calculadora Automática* – Receber dois números reais (`float`) e executar as quatro operações aritméticas de adição, subtração, multiplicação e divisão com os dois números. Em seguida, imprimir os resultados no monitor.