

Introdução à Programação



Introdução a Linguagem C

Construções Básicas

Programa em C

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

```
int main ( ) {
```

```
float celsius ;
```

```
float fahrenheit ;
```

```
celsius = 30;
```

```
fahrenheit = 9.0/5 *celsius + 32;
```

```
printf ("30 graus celsius = %f graus fahrenheit",fahrenheit);
```

```
return 0;
```

```
}
```

Palavras
Reservadas

E ainda
expressões...

Identificadores criados
por terceiros (chamadas
de funções)

Identificadores
criados pelo
programador

Tópicos da Aula

- ◆ Hoje aprenderemos a escrever um programa em C, para isto veremos
 - Características da linguagem
 - Estrutura de um programa
 - Ciclo de construção de um programa em C
 - Estruturas básicas de uma linguagem de programação
 - Identificadores, Variáveis, Tipos de dados, Atribuição
 - Conceito de funções
- ◆ Depois escreveremos um programa em um ambiente de programação
 - Apresentação de um ambiente de programação
 - Executando um programa

Linguagem C - Características

- ◆ Linguagem de programação de finalidade geral criada por Dennis Ritchie em 1972
 - Criada para desenvolver o sistema operacional Unix
 - Padronizada em 1989 pela ANSI
 - Tornou-se muito popular
- ◆ Ela faz parte das linguagens denominadas imperativas
 - Ações que o computador deve executar são especificadas como comandos que alteram o estado da memória

Linguagem C - Características

- ◆ Combina o *alto nível* com o *baixo nível*, permitindo a manipulação direta de bits, bytes e endereços de memória
- ◆ Possui fluxos de controle e estruturas de dados presentes na maioria das linguagens imperativas
 - Agrupamento de comandos
 - Tomadas de decisões (*if-else*)
 - Laços para permitir a repetição de comandos

Linguagem C - Características

- ◆ Possui apenas 32 palavras-chaves (reservadas)
- ◆ Permite economia de expressão e gera códigos reduzidos
- ◆ Possibilita alocação dinâmica de memória
- ◆ Permite estruturar o software em módulos, arquivos fontes, bibliotecas

Linguagem C - Críticas

- ◆ Dá-se muita liberdade ao programador
- ◆ Programas ininteligíveis, acesso direto à memória
- ◆ Não há verificação de tipos e nem de limites de *arrays* (*veremos depois*)
 - Simplifica o projeto do compilador C
- ◆ Mensagens de erro muito vagas (limitação do compilador)

Estrutura de um Programa

- ◆ Em C, a construção fundamental é a **função**:
 - Um programa é constituído de uma ou mais funções
 - Uma destas funções deve ser a função **main**
 - **main** é a função principal do programa
 - O programa inicia sua execução na função **main**

Mas, o que é uma função?

É um conjunto de instruções para realizar uma ou mais tarefas que são agrupadas em uma mesma unidade e que pode ser referenciada por um nome único

Usando o Conceito de Funções para Estruturar Programas

```
função soma {  
    instrução 1  
    instrução 2  
    ...  
}  
  
função mostraResultado {  
    instrução 1  
    instrução 2  
    ...  
}  
  
função principal {  
    soma  
    mostraResultado  
    instrução 3  
    ...  
}
```

Cada vez que a função
é chamada, as
instruções da função
são executadas

Estrutura de um Programa

- ◆ Dentro de uma função pode haver vários tipos de instruções:
 - Declarações, expressões, chamadas de outras funções, comandos, etc
- ◆ Uma função em C pode retornar algum valor, assim como acontece com funções matemáticas
 - Inteiro, real, caractere, etc
- ◆ Porém, uma função **não precisa** necessariamente retornar um valor
 - Quando não retorna um valor, dizemos que a função é do tipo ***void***

Estrutura de um Programa em C

função 1

função 2

.

.

.

função n

int main ()

{

.

.

.

}

Comandos,
declarações, chamada
de funções, etc

Exemplo de um Programa em C

```
#include <stdio.h>
int main()
{
    printf("Ola Mundo!");
}
```



Mostra na tela a mensagem
Ola Mundo!

Outro Exemplo de um Programa em C

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

```
int main()
```

```
{
```

```
    int resultado;
```

Declaração de variavel

```
    resultado = 6 + 7;
```

Expressão aritmética

```
    printf("%d", resultado);
```

```
    return 0;
```

Chamada de função

Comando

```
}
```

Estrutura de uma Função

Assinatura da função

```
int multiplicacao (int p1, int p2)
{
    int produto;
    produto = p1 * p2;
    return produto;
}
```

Corpo da
função

◆ Uma função deve conter:

- Uma assinatura
- Um corpo

Estrutura de uma Função

Tipo retornado

Nome

Lista de parâmetros

```
int multiplicacao (int p1, int p2)
{
    int produto;
    produto = p1 * p2;
    return produto;
}
```

Parênteses
obrigatórios

◆ **Assinatura** de uma função deve informar:

- Tipo de valor retornado (se for o caso)
- Nome
- Lista de parâmetros formais (se houver)
 - Nome do parâmetro e tipo do parâmetro

Estrutura de uma Função

```
int multiplicacao (int p1, int p2)
{
    int produto;
    produto = p1 * p2;
    return produto;
}
```

Corpo da
função
dentro das
chaves { }

Instruções devem ser separadas
por ponto e vírgula (;)

◆ **Corpo** de uma função contém:

- Instruções
 - Declarações, comandos, expressões, etc
- O comando **return**, caso o tipo de retorno seja **diferente** de **void**

Funções de Bibliotecas

- ◆ A linguagem C oferece um conjunto de funções já definidas que são organizadas em bibliotecas
 - Programador pode utilizar (chamar) estas funções no programa
 - Reduzem o tempo de desenvolvimento do programa
- ◆ Algumas das funções oferecidas permitem:
 - realizar operações de E/S
 - manipulação de caracteres (strings)
 - aplicações matemáticas, etc.

Funções de Bibliotecas

- ◆ Informações sobre as funções de bibliotecas são encontradas em vários arquivos
 - header files ou arquivos com extensão .h
- ◆ Para usar estas bibliotecas, o programa deve indicar os nomes dos arquivos que as contêm
- ◆ Estas bibliotecas são adicionadas ao programa usando a **diretiva de pré-processamento** `# include`
 - Por exemplo: `# include < stdio.h >`
 - Não possui ponto_e_vírgula (;)
 - **stdio.h** é uma biblioteca que contém funções de entrada/saída, tal como a função **printf** que permite mostrar alguma mensagem no monitor

Outro Exemplo de Programa em C

```
#include <stdio.h>
int multiplicacao (int p1, int p2)
{
    int produto;
    produto = p1 * p2;
    return produto;
}
int main ( )
{
    int resultado;
    resultado = multiplicacao (6,7) ;
    printf("6 vezes 7 eh %d", resultado) ;
    return 0 ;
}
```

Inclui a biblioteca stdio

Chama a função multiplicacao

Argumentos da função

Chama função da biblioteca stdio

Diretivas de Pré-Processamento

- ◆ O **pré-processador** é um programa que examina o programa fonte em C e executa certas modificações com base em instruções chamadas de **diretivas**
 - Exemplos: ***include***, ***define***, etc
- ◆ Uma diretiva deve vir precedida de **#**
 - Deve ser escrita em uma única linha
 - Se passar de uma linha deve-se usar a barra invertida (\) ao final da linha e continuar na seguinte
- ◆ Diretivas não fazem parte da linguagem C
 - Servem para auxiliar o desenvolvimento do programa fonte

Exemplo da Diretiva include

```
#include <stdio.h>
int multiplicacao (int p1, int p2)
{
    int produto;
    produto = p1 * p2;
    return produto;
}
int main ( )
{
    int resultado;
    resultado = multiplicacao(6,7);
    printf("6 vezes 7 eh %d",resultado);
    return 0 ;
}
```

Inclui a biblioteca stdio

Chama função da biblioteca stdio

Variáveis

- ◆ Programas manipulam dados (valores) e esses dados são armazenados em ***variáveis***
- ◆ Uma variável é uma posição na memória referenciada por um identificador (nome)
- ◆ Uma variável deve ser ***declarada*** informando o tipo de dado que ela armazenará e o nome dela

tipo nome
 ↘ ↙
 int total;
 int count, temp, result;

Muitas variáveis podem ser criadas em uma declaração

Variáveis

- ◆ Uma **declaração** de variável instrui o compilador:
 - a reservar um espaço de memória suficiente para armazenar o tipo de dado declarado
 - o nome ao qual iremos referenciar esta posição de memória
- ◆ Só após a declaração da variável, é que esta pode ser referenciada (utilizada)
- ◆ Quando uma variável é referenciada no programa, o valor armazenado nela é utilizado

Variáveis

- ◆ Onde se declara variáveis?
 - No corpo de uma função (variável local)
 - Recomenda-se que as declarações de variáveis sejam as primeiras instruções da função
 - Na lista de parâmetros de funções
 - Parâmetros formais
 - Fora das funções (variável global)

Declaração de Variáveis

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

```
int resultado;
```

Declaração de
variável global

Declaração de variáveis nos
parâmetros formais

```
int multiplicacao (int p1, int p2)
```

```
{
```

```
    int produto;
```

Declaração de variável local

```
    produto = p1 * p2;
```

```
    return produto;
```

```
}
```

```
int main () {
```

```
    resultado = multiplicacao(6,7);
```

```
    printf("6 vezes 7 eh %d", resultado);
```

```
    return 0 ;
```

```
}
```

Atribuição

- ◆ Um **comando** de **atribuição** modifica o valor armazenado na variável
- ◆ O operador de atribuição é o sinal de =

```
int total;
```

```
total = 55;
```

Variável *total* armazena valor 55



Valor 65 sobrescreve o valor armazenado antes

```
total = 65;
```




Só se deve atribuir a uma variável valores compatíveis com o tipo declarado da variável

Inicialização de Variáveis

- ◆ Uma variável pode ser inicializada com o comando de atribuição na hora de sua declaração

Declara variável *total* do tipo `int`

`int total = 55;` Inicializa *total* com valor 55



Em C, variáveis locais não são inicializadas automaticamente

Constantes

- ◆ Uma **constante** é um identificador semelhante a uma variável, exceto pelo fato de só poder armazenar o mesmo valor durante toda sua existência
- ◆ Uma constante é declarada usando a palavra reservada **const**
- ◆ Deve-se inicializar a constante no ato da sua declaração

```
const float PI = 3.1416;
```

```
PI = 3.141618;
```

Esse comando gera um
erro de compilação

Não se pode mudar o valor de uma
constante

Constantes

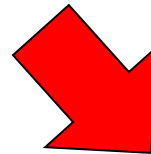
- ◆ São úteis para dar um significado mais compreensível a determinados valores
 - Exemplo : π é mais compreensível que o valor 3,1416
- ◆ Facilitam a manutenção do programa
 - Caso uma alteração no programa seja necessária que acarrete uma mudança no valor da constante e esta constante seja referenciada em vários lugares do programa, só precisamos alterar o programa em um lugar
- ◆ Explicitam formalmente que um determinado valor não pode ser alterado
 - Evitam erros de outros programadores

Constantes

- ◆ É comum utilizar-se também a diretiva de pré-processamento **define** para definir constantes

```
#define PI 3.1416
int main() {
    float raio = 5;
    float area = PI*raio*raio;
    float comp = 2*PI*raio;
    return 0;
}
```

No pré-processamento, ocorre a substituição



```
int main() {
    float raio = 5;
    float area = 3.1416*raio*raio;
    float comp = 2*3.1416*raio;
    return 0;
}
```

Linguagem C - Tipos de Dados

- ◆ Cada dado possui associado a ele um **tipo** e pode possuir um **qualificador**
- ◆ C possui 5 tipos básicos de dado
 - char: tipo caractere (tamanho de um byte)
 - int: tipo inteiro (números sem parte decimal)
 - float: tipo ponto flutuante de precisão simples
 - double: tipo ponto flutuante de precisão dupla
 - void: não possui valor
 - Mais utilizado para indicar que uma função não retorna nenhum valor
- ◆ Qualificadores: short, long, unsigned, signed
 - Precedem o tipo na declaração do tipo
 - Ex: `unsigned int valor ;`

Linguagem C - Tipos de Dados

- ❖ O tipo de dado define o tamanho do dado e a forma de armazenamento

Valores Inteiros e suas Representatividades

Tipo	Tamanho	Representatividade
char	1 byte	-128 a 127
unsigned char	1 byte	0 a 255
short int	2 bytes	-32768 a 32767
unsigned short int	2 bytes	0 a 65535
long int (ou int) em ambientes de 32 bits	4 bytes	-2147483648 a 2147483647
unsigned long int	4 bytes	0 a 4294967295

Linguagem C - Tipos de Dados

Valores Reais e suas Representatividades

Tipo	Tamanho	Representatividade
float	4 bytes	$\pm 3.4 \times 10^{-38}$ a 3.4×10^{38}
double	8 bytes	$\pm 1.7 \times 10^{-308}$ a 1.7×10^{308}
long double	10 bytes	$\pm 3.4 \times 10^{-4932}$ a 3.4×10^{4932}

Tipos de Dados Numéricos

◆ Números com e sem sinal

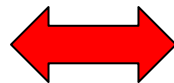
- C permite que o programador defina se uma variável de tipo numérico deva ou não reservar o bit de sinal (números negativos)
- Notação
 - **signed tipo**
 - **unsigned tipo**
- Se nenhum modificador for indicado, o compilador C reservará o bit de sinal

Tipo de Dados Caractere

◆ Representado pelo tipo **char**

- 'a', 'b', '1', '\n' etc
- internamente representa um código da tabela ASCII
- ASCII: na verdade aceita até 255 caracteres (unsigned char)

```
char letra = 'A' ;
```



```
char letra = 65 ;
```

Instrução equivalente

Identificadores

- ◆ Identificadores são palavras que o programador utiliza em programas
- ◆ Consiste de um ou mais caracteres.
 - o primeiro caractere deve ser letra ou "_"
 - demais são uma combinação de letras, números e "_"
 - Identificadores não podem começar com um dígito
 - Exemplo de identificadores **válidos**: _a, a3_, bom_dia
 - Exemplo de identificadores **inválidos**: 2a, a-b, a_ b
 - C é "case-sensitive"
 - Os identificadores **casa** e **CASA** são diferentes

Identificadores

- ◆ Identificadores podem ser:
 - Nomes que o programador escolheu
 - Exemplo: nome de uma variável, função, constante, etc
 - Nomes que terceiros escolheram
 - Exemplo: nome de uma função de uma biblioteca utilizada
 - Palavras reservadas da linguagem
 - Não podem ser usadas de outra forma
 - Exemplo: *main*, *const*, *int*, etc

Identificadores em C

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

```
int main ( ) {
```

```
float celsius ;
```

```
float fahrenheit ;
```

```
celsius = 30;
```

```
fahrenheit = 9.0/5 *celsius + 32;
```

```
printf ("30 graus celsius = %f graus fahrenheit",fahrenheit);
```

```
return 0;
```

```
}
```

Palavras
Reservadas

E ainda
expressões...

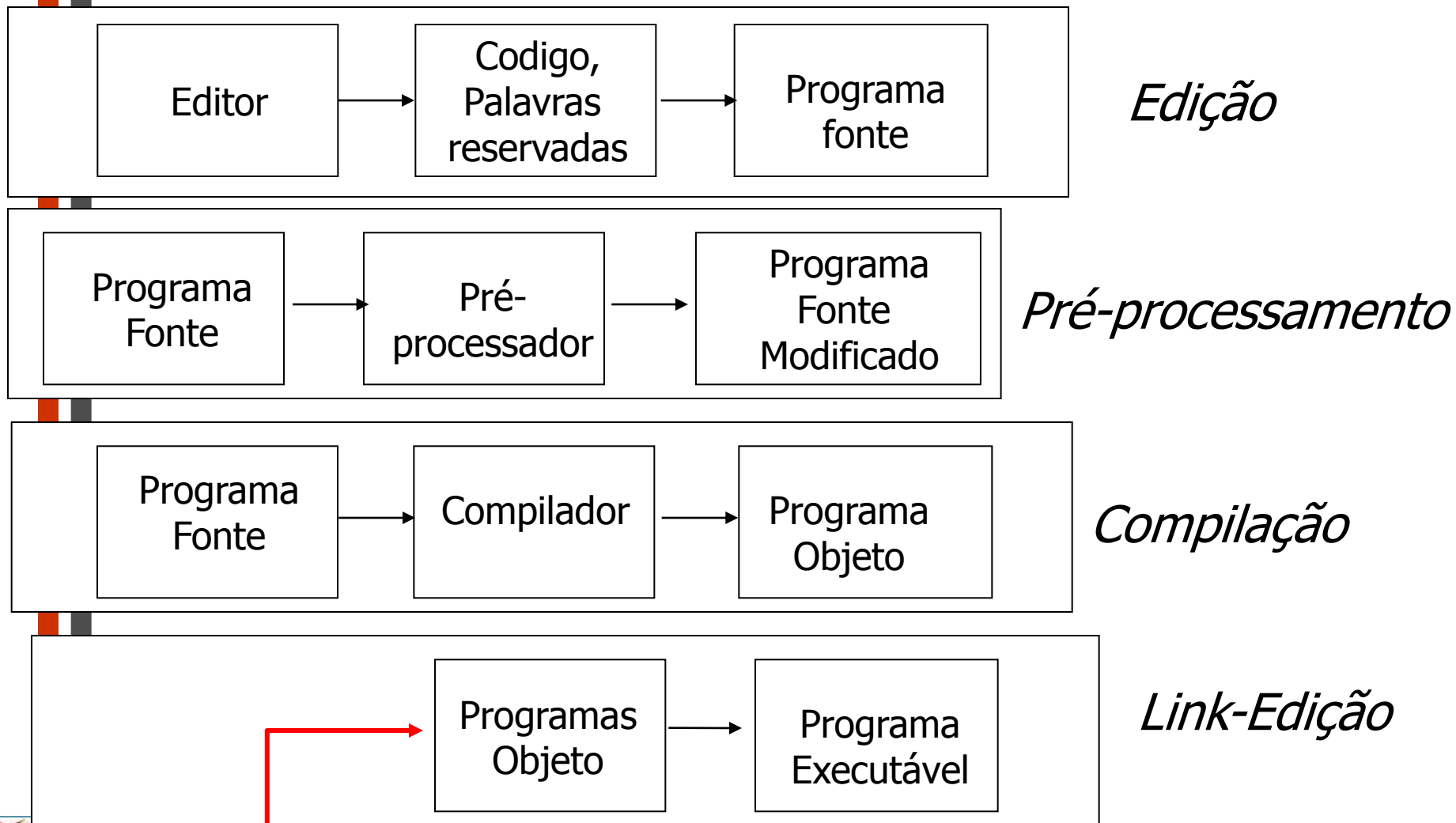
Identificadores criados
por terceiros (chamadas
de funções)

Identificadores
criados pelo
programador

Executando de um Programa em C

- ◆ Escrever o programa em um arquivo texto (código fonte)
 - Salvar o arquivo com a extensão .c
- ◆ Compilar o programa fonte para gerar o código executável (*.exe)
 - Um programa em C pode ser composto de vários códigos fontes (vários arquivos .c)
 - É comum a geração de um código objeto (*.obj ou *.o) para cada código fonte e a posterior geração do código executável (linkedição)

Construção de Programas em C



Adição /Localização de códigos objetos das bibliotecas

Ambiente de Programação

- ◆ Para acelerar o desenvolvimento de programas, é comum utilizar ambientes de programação ou **IDEs** (Integrated Development Environment)
 - Integra várias ferramentas em um único ambiente
 - Editores de texto
 - Compiladores
 - Bibliotecas
 - E muito mais ...
 - **CodeBlocks, DevC++, Visual Studio, Eclipse etc**

Resumindo ...

- ◆ Características da linguagem
- ◆ Estrutura de um programa
- ◆ Ciclo de construção de um programa em C
- ◆ Estruturas básicas de uma linguagem de programação
 - Identificadores, Variáveis, Tipos de dados, Atribuição
- ◆ Conceito de funções