Construção de Algoritmos e Programação

Aula 9 - 28/04/2025

Joice Otsuka
joice@ufscar.br

Programação C - Conceitos Básicos

Programação

- Neste curso, será utilizada a linguagem C
- Utilizaremos um ambiente integrado de desenvolvimento (IDE):
  - Editor de programas: viabiliza a escrita do programa.
  - Compilador: verifica se o texto digitado obedece à sintaxe da linguagem de programação e, caso isto ocorra, traduz o texto para uma sequência de instruções em linguagem de máquina.
- Exemplos de IDE:
  - GDB online - https://www.onlinegdb.com/
  - Visual Studio
  - Code::Blocks

Escrevendo um programa em C

Primeiro exemplo

- Considere o programa p1.c

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

int main()
{
    float y;
    y=sin(1.5);
    printf("y = %f",y);
    printf("\n");
    return 0;
}
```

Arquivos de cabeçalho

- Note que o programa-fonte p1.c começa com as linhas:

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

- Todo programa-fonte em linguagem C começa com linhas deste tipo.
  - Informam ao compilador que o programa-fonte vai utilizar arquivos de cabeçalho (extensão .h, de header).
  - Arquivos de cabeçalho contém informações que o compilador precisa para construir o programa executável.

Arquivos de cabeçalho

- Observe que o programa p1.c utiliza algumas funções, tais como:
  - sin – função matemática seno.
  - printf – função para exibir resultados.

- Por serem muito utilizadas, a linguagem C mantém funções como estas em bibliotecas.

Arquivos de cabeçalho

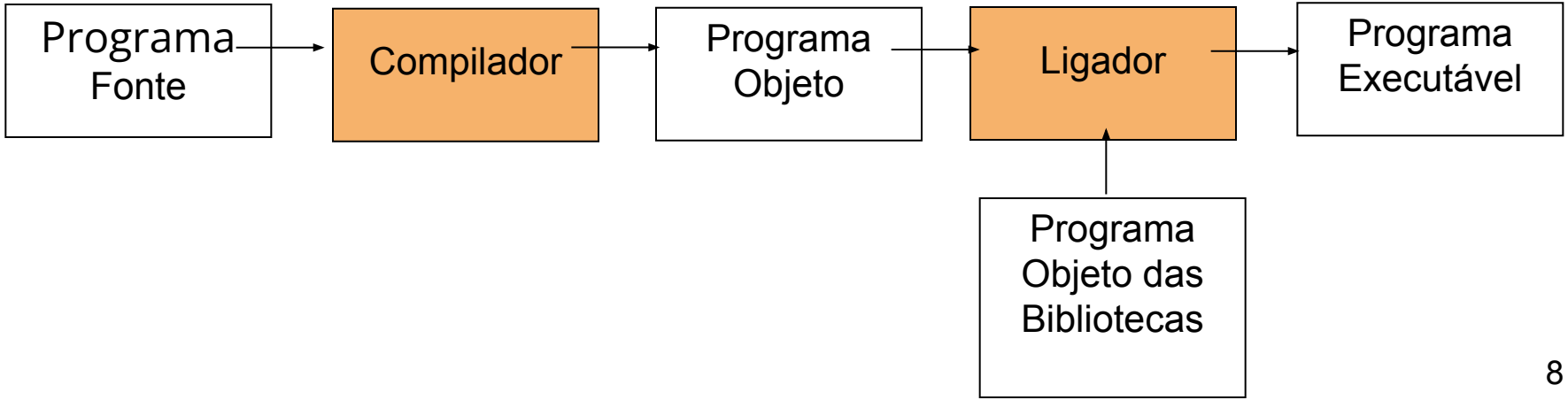
- Portanto, as linhas:

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

indicam ao compilador que o programa p1.c utilizará as instruções das bibliotecas stdio e stdlib.

Processo de compilação

- O processo de compilação se dá em duas etapas:
  - Fase de tradução: programa-fonte é transformado em um programa-objeto.
  - Fase de ligação: junta o programa-objeto às instruções necessárias das bibliotecas para produzir o programa executável.



Função main

- Todo programa C contém uma função main
- O corpo de uma função sempre começa com abre-chaves { e termina com fecha-chaves }.

Corpo da função →

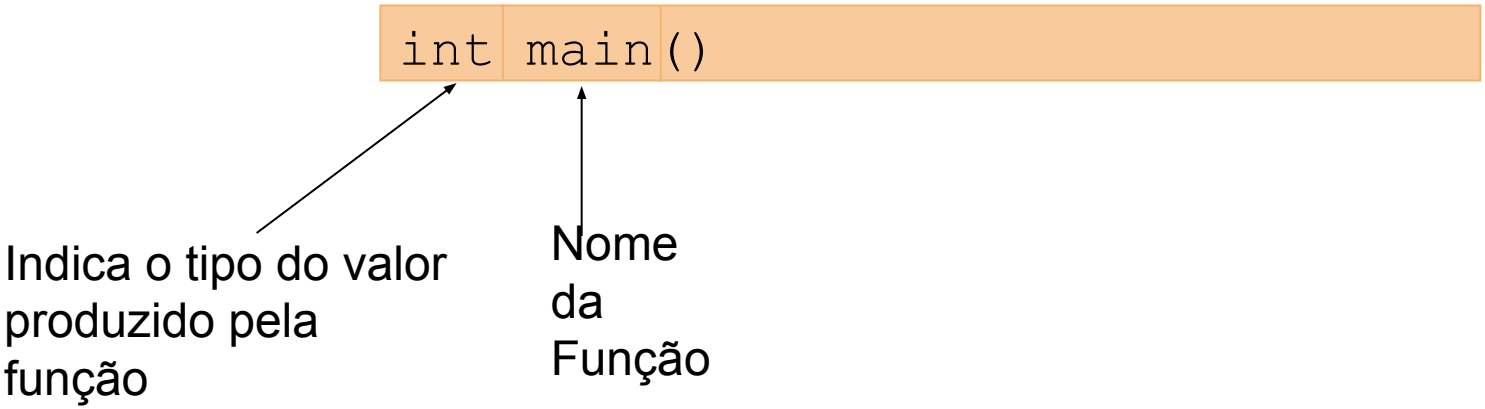
```c
#include <stdio.h>
#include <math.h>

int main()
{
    float y;
    y=sin(1.5);
    printf("y = %f",y);
    printf("\n");
    return 0;
}
```

Função main

```
int main()
```

- Esta linha corresponde ao cabeçalho da função main (a função principal).
- O texto de um programa em Linguagem C pode conter muitas outras funções e **SEMPRE deverá conter a função main.**



Função main

- A linguagem C é *case sensitive*. Isto é, considera as letras maiúsculas e minúsculas diferentes.
  - O nome da função principal deve ser escrito com **letras minúsculas: main.**
  - Main ou MAIN, por exemplo, provocam erros de sintaxe.

Variáveis

- Os dados que um programa utiliza precisam ser armazenados na memória do computador.
- Cada posição de memória do computador possui um endereço.
- Para facilitar o controle sobre onde armazenar informação, os programas utilizam variáveis
- Uma variável corresponde a um nome simbólico de uma posição de memória
- Seu conteúdo pode variar durante a execução do programa

Variáveis

- Dentro do programa, as variáveis são identificadas por seus nomes.
- Portanto, um programa deve declarar todas as variáveis que irá utilizar.
- **Atenção!**
  - **A declaração de variáveis deve ser feita antes que a variável seja usada**, para garantir que a quantidade correta de memória já tenha sido reservada para armazenar seu valor.

Declaração de variáveis

- **Na linguagem C toda variável deve ser declarada antes de ser usada**
  - tipo nome;

- Ex:
  - int x; // inteiro
  - float y; // real
  - char z; // caracter
  - int a,b,c;

Declaração de Variáveis

- O nome de uma variável deve ser único e deve seguir algumas regras:
  - ser sucinto e utilizar nomes significativos
  - não utilizar espaços entre as letras
    - nome do cliente (errado!)
    - o correto é nome_do_cliente, nomeCliente
  - não iniciar o nome da variável com números
    - 2valor (errado)
    - o correto é valor2
  - não utilizar caracteres especiais, como símbolos (? / : @ #)
  - não utilizar palavras reservadas

Declaração de Variáveis

- Cada variável pode possuir uma quantidade diferente de bytes, uma vez que os tipos de dados são representados de forma diferente.

O tipo da variável define quantos bytes de memória serão necessários para representar os dados que a variável armazena.

Tipos de dados básicos

- A Linguagem C dispõe de quatro tipos básicos de dados. Assim, as variáveis poderão assumir os seguintes tipos:

Tipo	Bits	Intervalo de valores
char	8	-128 A 127
int	32	-2.147.483.648 A 2.147.483.647
float	32	1,175494E-038 A 3,402823E+038
double	64	2,225074E-308 A 1,797693E+308

Declaração de variável

- A primeira linha do corpo da função principal do programa **p1.c** é uma declaração de variável:

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

int main()
{
    float y;
    y=sin(1.5);
    printf("y = %f",y);
    printf("\n");
    return 0;
}
```

Declaração de variável

```
float y;
```

- Esta linha declara uma variável y para armazenar um número de ponto flutuante.

- **A declaração de uma variável não armazena valor algum na posição de memória que a variável representa.**

- Ou seja, no caso anterior, vai existir uma posição de memória chamada y, mas ainda não foi armazenado valor nessa posição (pode ter “lixo”)

Atribuição de valor

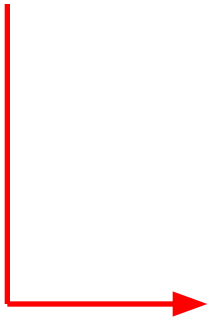
- Um valor pode ser atribuído a uma posição de memória representada por uma variável pelo operador de atribuição **=**
- O operador de atribuição requer à esquerda um nome de variável e à direita, um valor

Exemplos:

```
float y;
int x;
y = 1.5;
x = 5;
```

Atribuição de valor

- A linha seguinte de p1.c atribui um valor a y:



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

int main()
{
    float y;
    y=sin(1.5);
    printf("y = %f",y);
    printf("\n");
    return 0;
}
```

- No lado direito do operador de atribuição existe uma referência à função sin (seno) com um parâmetro 1.5

Atribuição de valor

- Em uma linguagem de programação chamamos o valor entre parênteses da função, neste exemplo, o valor 1.5, de **parâmetro da função**.

- Da mesma forma, diz-se que sin(1.5) é o valor da função sin para o parâmetro 1.5.

- O operador de atribuição na linha y = sin (1.5) obtém o valor da função (0.997495) e o armazena na posição de memória identificada pelo nome y

Atribuição de valor

- **Atenção: O valor armazenado em uma variável por uma operação de atribuição depende do tipo da variável.**

- Se o tipo da variável for int, será armazenado um valor inteiro (caso o valor possua parte fracionária, ela será desprezada).

```
int y;
y = 3.4; // o valor 3 é armazenado em y
```

- Se o tipo da variável for float ou double, será armazenado um valor de ponto flutuante (caso o valor não possua parte fracionária, ela será nula).

```
float x;
x = 3;  // o valor 3.0 é armazenado em x
```

Atribuição de valor

- Considere as seguintes declarações:

```
int a;
float b;
```

- Neste caso, teremos:

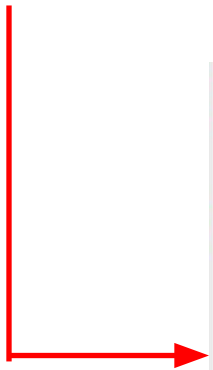
Operação de atribuição	Valor armazenado
a = (2 + 3) * 4	20
b = (1 – 4) / (2 - 5)	1.0
a = 2.75 + 1.12	3
b = a / 2.5	1.2

Saída de dados

Saída de dados

- As próximas linhas do programa p1.c são:

```
printf(“y = %f”,y);
printf(“\n”);
```



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

int main()
{
    float y;
    y=sin(1.5);
    printf("y = %f",y);
    printf("\n");
    return 0;
}
```

- A função **printf** faz parte da biblioteca **stdio**.

Saída de dados

- A função **printf** é usada para exibir resultados produzidos pelo programa e pode ter **um ou mais parâmetros**.

- O **primeiro parâmetro** da função printf é sempre uma **string**, correspondente à sequência de caracteres que será exibida pelo programa.
- String é um conjunto de caracteres delimitado por “ ”

```
printf(“y = %f”,y);
printf(“\n”);
```

Saída de dados

- Essa sequência de caracteres pode conter algumas tags que representam valores. Essas tags são conhecidas como especificadores de formato.

```
printf(“y = %f”,y);
printf(“\n”);
```

- Um **especificador de formato** começa sempre com o símbolo **%**. Em seguida, pode apresentar uma letra que indica o tipo do valor a ser exibido.

Especificadores de formato

%c	caracter
%d ou %i	inteiros
%u	inteiro sem sinal
%f	float
%lf	double
%Lf	long double
%s	String (conjunto de caracteres)
%ld	long int
%e ou %E	float ou double em notação científica
%%	%

Saída de dados

```
printf(“y = %f”,y);
printf(“\n”);
```

- Na função printf, **para cada tag** existente no primeiro parâmetro, deverá haver um novo parâmetro que especifica o valor a ser exibido.

- Assim, **printf(“y = %f”,y)** irá exibir a letra y, um espaço em branco, o símbolo =, um espaço em branco, e o valor armazenado na variável y em ponto flutuante.

Saída de dados

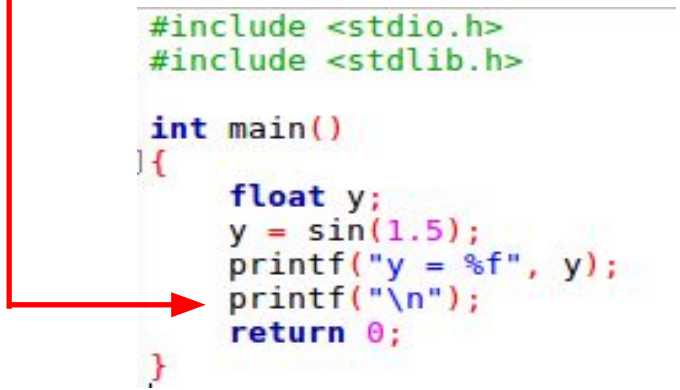
- A linguagem C utiliza o símbolo \ (barra invertida) para especificar alguns caracteres especiais:

Caractere	Significado
\n	Caractere (invisível) de nova linha.
\t	Caractere (invisível) de tabulação horizontal.
\’	Caractere de apóstrofo

Saída de dados

- Observe a próxima linha do programa p1.c:

printf(“\n”);



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

int main()
{
    float y;
    y = sin(1.5);
    printf("y = %f", y);
    printf("\n");
    return 0;
}
```

- Ela exibe o caractere (invisível) de nova linha. Ou seja, provoca uma mudança de linha

Formatação de valores numéricos

- Uma tag pode especificar o número total de caracteres (incluindo o sinal e o ponto decimal).

- Assim, a tag %8.3f significa: “exibir um valor de ponto flutuante com oito caracteres no total e com três casas decimais”.

- Se for necessário, será acrescentado o caractere ‘ ‘ (espaço) à esquerda do valor para completar o tamanho total.

Formatação de valores numéricos

- Exemplo:

Valor	Tag	Valor exibido
pi = 3.14159	%5.3f	3.142
	%8.3f	3.142
raio = 2.0031	%5.3f	2.003
	%.6f	2.003100
area = 2*pi*raio	%5.3f	12.586
	%6.3f	12.586
	%7.3f	12.586

Formatação de valores numéricos

A formatação de valores pode ser feita também para números inteiros.

Exemplo:

Valor	Tag	Valor exibido
3	%d	3
	%5d	3
	%01d	3
	%05d	00003

Entrada de dados

Leitura de dados

- A leitura de dados (entrada de teclado) em C é realizada por meio da função **scanf.**

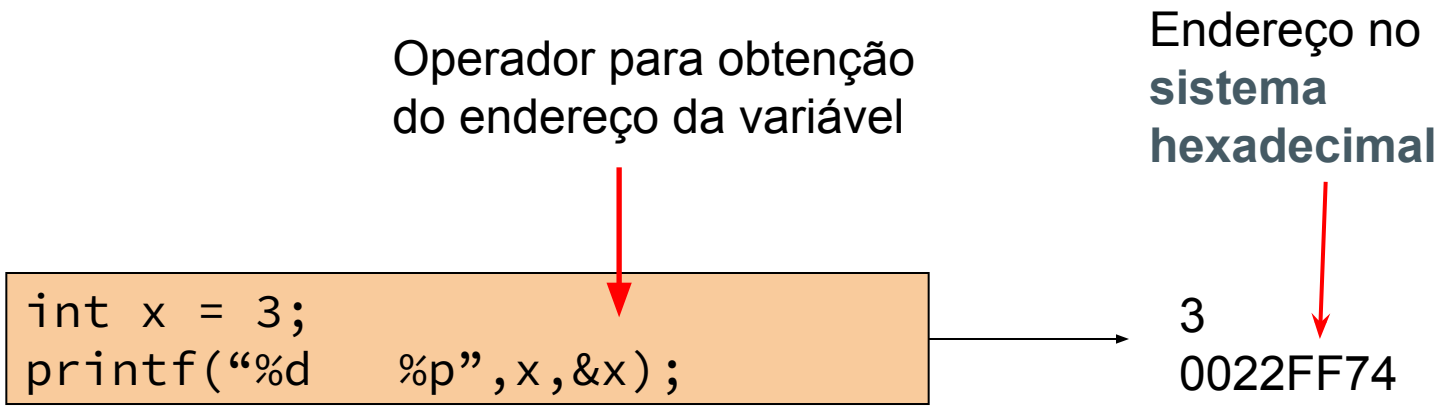
- Assim como printf, a função scanf pode ter vários parâmetros, sendo o primeiro uma string. No caso da função scanf, esta string deve **conter apenas tags, separadas por espaços em branco**.

- Os demais parâmetros da função scanf devem ser **endereços de variáveis** (referência/ponteiro para a variável).

```
scanf("%d %f",&codigo, &valor);
```

Endereços de variáveis

- Uma **variável** representa um **nome simbólico** para uma posição de memória.
- Cada posição de memória de um computador possui um **endereço**. Logo, o endereço de uma variável é o endereço da posição de memória representada pela variável.
- A tag usada para exibir endereços de memória é **%p**.



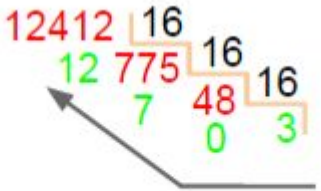
Decimal	Binário	Hexadecimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Cada dígito hexadecimal representa 4 bits

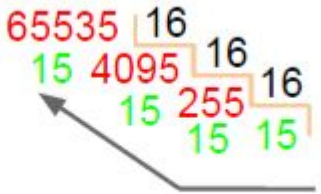
Exemplo:

Decimal: 79
Binário: 0100 1111
Hexadecimal: 4 F

Sistema hexadecimal



12412 Decimal = 307C Hexadecimal



65535 Decimal = FFFF Hexadecimal

Leitura de dados

- O que acontece quando o computador executa uma instrução de leitura de dados? Exemplo:

```
scanf("%f",&C);
```

- A execução do programa é interrompida até que o usuário digite algum valor e pressione a tecla Enter.

- Após pressionar Enter, o computador retoma a execução do programa e armazena o(s) valor(es) digitado(s), no(s) respectivos endereço(s) de variáveis fornecido(s) na função scanf.

Especificadores de formato

SPECIFIER	USED FOR
%c	a single character
%s	a string
%hi	short (signed)
%hu	short (unsigned)
%Lf	long double
%n	prints nothing
%d	a decimal integer (assumes base 10)
%i	a decimal integer (detects the base automatically)
%o	an octal (base 8) integer
%x	a hexadecimal (base 16) integer
%p	an address (or pointer)
%f	a floating point number for floats
%u	int unsigned decimal
%e	a floating point number in scientific notation
%E	a floating point number in scientific notation
%%	the % symbol

Leitura de dados

- Em programação, diz-se que coisas são **estáticas** quando ocorrem antes do programa executar e **dinâmicas**, quando ocorrem durante a execução.

```
C = 32;
```

Valor de C é estabelecido **estaticamente**.

```
scanf("%f",&C);
```

Valor de C é estabelecido **dinamicamente**.

Leitura de dados

- Na leitura de dados, o valor digitado pelo usuário deve ser do **mesmo tipo** que a variável.

- Com a leitura de dados, a execução de um programa pode ser realizada para valores diferentes das variáveis.

- Porém, se o valor da variável é estabelecido de forma estática, para cada troca de valor da variável é necessário compilar o programa novamente.

Exercício

- Dada uma temperatura em graus Celsius apresentá-la em graus Fahrenheit. A fórmula de conversão é:

  F = (9 * C + 160) / 5

```c
#include <stdio.h>

int main()
{
    float C, F;
    printf("Informe a temperatura em graus Celsius: ");
    scanf("%f", &C);
    F = (9 * C + 160) / 5;
    printf("A temperatura informada corresponde a %.1f graus F\n", F);
    return 0;
}
```

Erros de sintaxe

- O programa executável só será gerado se o texto do programa não contiver erros de sintaxe (rigidez de sintaxe).
- Exemplo: considere o comando printf abaixo:

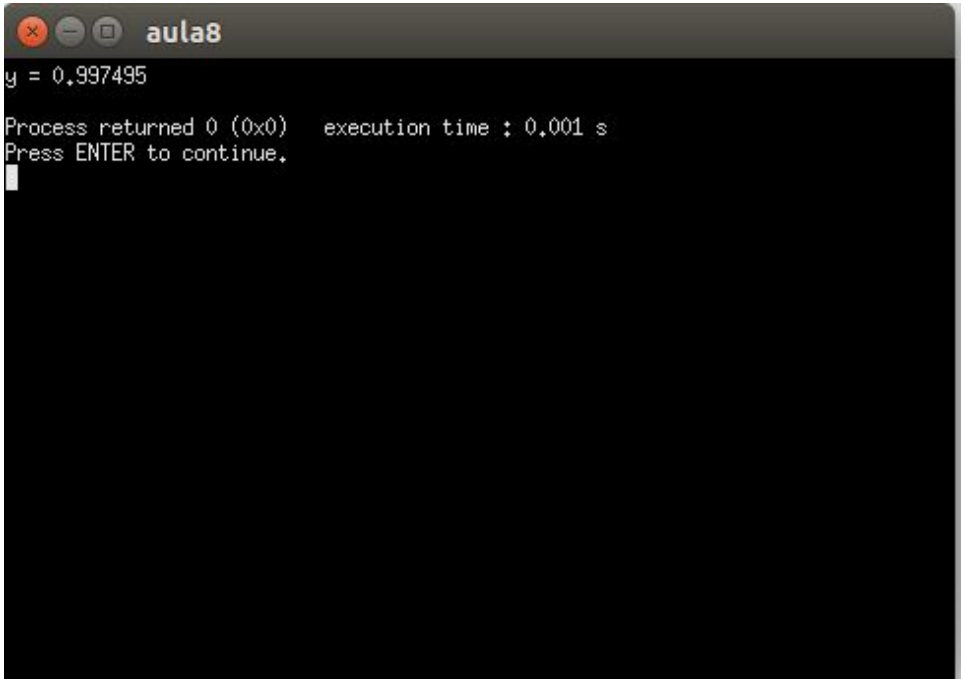
```
printf(“y = %f,y);
```

  - Como faltou fechar as aspas no primeiro parâmetro do printf (que deveria ser uma string), o compilador apontará um erro de sintaxe nesta linha do programa, exibindo uma mensagem tal como:

```
undetermined string or character constant
```

Erros de sintaxe

- Se o nome do programa é p1.c, então após a compilação, será produzido o programa executável p1.exe.
- Executando-se o programa p1.exe, o resultado será:



Tipos de dados, modificadores e conversão implícita e explícita

Tipos de dados básicos

- Como visto, a Linguagem C dispõe de quatro tipos básicos de dados:

Tipo	Bits	Intervalo de valores
char	8	-128 A 127
int	32	-2.147.483.648 A 2.147.483.647
float	32	1,175494E-038 A 3,402823E+038
double	64	2,225074E-308 A 1,797693E+308

Modificadores de tipo

- A linguagem C define alguns modificadores de tipo.
- Um modificador de tipo altera o intervalo de valores que uma variável pode armazenar.
  - **short:** aplicável ao tipo **int**.
    - uma variável **short int** tem 16 bits (2 bytes)
    - pode armazenar valores inteiros no intervalo: -2^{15} a $2^{15} - 1$
  - **long:** aplicável aos tipos **int** e **double**
    - um **long int** tem 64 bits (8 bytes)
    - um **long double** terá maior precisão (16 bytes)
  - **unsigned:** a variável terá apenas valores positivos (char ou int)

Modificadores de tipo

- Para as variáveis do tipo **char**, o compilador reserva 8 bits. Assim, variáveis do tipo **char** podem armazenar valores inteiros no intervalo -2^7 a $2^7 - 1$.

- O modificador de tipo **unsigned** instrui o compilador a não considerar o primeiro bit como sinal. Assim, variáveis **unsigned char** podem representar valores positivos maiores. O maior valor será: $2^8 - 1$.

Modificadores de tipo

- Resumindo
  - Ao tipo **float** não se aplica nenhum dos modificadores
  - Ao tipo **double** aplica-se apenas o modificador **long**
  - Ao tipo **char** aplica-se somente o tipo **unsigned**
  - Ao tipo **int** aplicam-se todos os modificadores (**short, long, unsigned)**

Representação de números inteiros

- Existem várias maneiras de representar números inteiros no sistema binário.
- Forma mais simples é a **sinal-magnitude:**
  - O bit mais significativo corresponde ao sinal e os demais correspondem ao valor absoluto do número.
- Exemplo: considere uma representação usando cinco dígitos binários (ou bits).

Decimal	**Binário**
+5	00101
-3	10011

Desvantagens:

- Duas notações para o zero (+0 e -0).
- A representação dificulta os cálculos.

```
  00101
  10011
Soma 11000
```

5 – 3 = – 8 ???

Representação de números inteiros

- A representação predominantemente assumida pelos computadores é a chamada **complemento-de-2:**
  - Para números positivos, a representação é idêntica à da forma sinal-magnitude.
  - Para os números negativos, a representação se dá em dois passos:
    1. Inverter os bits 0 e 1 da representação do número positivo;
    2. Somar 1 ao resultado.
  - Exemplo:

Decimal	**Binário**	
+6	00110	
-6	11001	(bits invertidos)
	1	(somar 1)
	11010	

Representação de números inteiros

- Note o que ocorre com o zero:

Decimal	Binário	
+0	00000	
-0	11111	(bits invertidos)
	1	(somar 1)
	00000	

Note que o ***vai-um*** daqui não é considerado, pois a representação usa apenas 5 bits.

- E a soma?

Decimal	Binário
+5	00101
-3	11101

Somando:

00101
11101
00010 ← Que corresponde ao número +2!

Overflow

Nesse programa são atribuídos os maiores valores possíveis às variáveis x e y.

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

int main(int argc, char *argv[])
{
  int x;
  short int y;
  char a;
  unsigned char b;

  x = pow(2,31)-1;     // maior int possível
  y = pow(2,15)-1;     // maior short int possível
  printf("x = %d  y = %d\n",x,y);
  x = x + 1;
  y = y + 1;
  printf("x = %d  y = %d\n",x,y);
  /* ------------
     Atribuir os maiores valores possíveis
     para as variáveis a e b.
     ------------------------------------ */
  a = pow(2,7)-1;
  b = pow(2,8)-1;
  printf("a = %d  b = %d\n",a,b);
  a = a + 1;
  b = b + 1;
  printf("a = %d  b = %d\n",a,b);
  system("PAUSE");
  return 0;
}
```

Overflow

- Em seguida, os valores das variáveis são incrementados de 1.
- O que acontece?
  - Ocorre um extravasamento (overflow)!

Exemplo: considere a variável y.

```
y = pow(2,15) - 1
```

```
0111 1111 1111 1111          32767
                1
----------------------------
1000 0000 0000 0000          -32768
```

Números de ponto flutuante

- A representação de ponto flutuante em binário tem três partes: o **sinal,** o **expoente** e a **mantissa**.
- A mantissa é representada na forma normalizada, ou seja, na forma **1.f**
- Ou seja, o primeiro bit sempre é 1, e os demais bits são representados em **f**.
- Exemplo 1:

Decimal	**Binário**	**Binário normalizado**
+13.25	1101.01	1.10101×2^3

Mantissa

Expoente

Números de ponto flutuante

- Exemplo 2:

Decimal	Binário	Binário normalizado
+0.25	0.01	1.0×2^{-2}

Mantissa

Expoente

- Existem dois formatos importantes para os números de ponto flutuante:
  - Precisão simples (SP) - Tipo **float** em C
  - Precisão dupla (DP) - Tipo **double** em C

Números de ponto flutuante

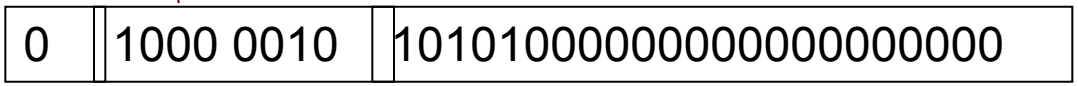
- Precisão Simples
  - Ocupa 32 bits: 1 bit de sinal, 8 bits para o expoente (representado na notação excesso-de-127) e 23 bits para a mantissa.
  - Exemplo:

Ponto flutuante

1.10101×2^{3}

127 + 3 = 130

Representação SP

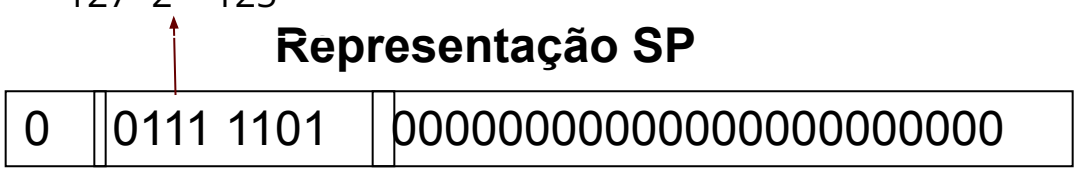


Ponto flutuante

1.0×2^{-2}

127 -2 = 125

Representação SP



  - O primeiro bit da mantissa de um número de ponto flutuante não é representado (sempre 1).

Números de ponto flutuante

- Precisão Dupla

  Ocupa 64 bits: 1 bit de sinal, 11 bits para o expoente (representado na notação excesso-de-1023) e 52 bits para a mantissa

- Exemplos: Similares aos abordados para precisão simples...

Avaliação de expressões aritméticas

- Os operadores aritméticos disponíveis na linguagem C são:

Operador	Operação
+	soma
-	subtração
*	multiplicação
/	divisão
%	resto da divisão

Conversão implícita de tipo

- Na avaliação de expressões aritméticas, estas **operações são realizadas sempre entre operandos do mesmo tipo**.

- Ou seja, o resultado da operação terá o mesmo tipo que os operandos.

- Caso haja valores inteiros e em ponto flutuante em uma expressão, haverá uma **conversão implícita de tipo** de **int** para **float**, sempre que necessário.

Prioridade de execução das operações

- Porque as operações aritméticas devem ser feitas entre operandos do mesmo tipo?
  - As representações dos números inteiros e dos números de ponto flutuante são diferentes.
- Ou seja, embora 1 e 1.0 sejam valores iguais, eles têm representações diferentes no computador.

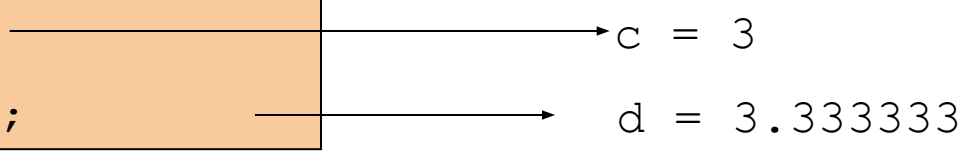
- Prioridade de execução das operações:

  1) expressões entre parênteses
  2) multiplicação, divisão e resto da divisão (da esquerda para a direita)
  3) operações de soma e subtração (da esquerda para a direita).

Conversão explícita de tipos

- É preciso **muito cuidado com a divisão inteira** (divisão entre operandos inteiros).
- O resultado da divisão entre operandos inteiros é sempre um número inteiro.
- Assim, se necessário, pode-se usar uma **conversão explícita de tipo** (*type casting*) para se obter um resultado float.

```
int a = 10, b = 3;
int c;
float d;
c = a / b;
d = (float) a / b;
```



Conversão explícita de tipos

- Atenção!
  - Observe que os resultados de:

```
d = (float) a / b;
```

(1)

e

```
d = (float)(a / b);
```

(2)

são diferentes!

  - Em (1), primeiro realiza-se primeiro a conversão explícita de tipo (a torna-se 10.0) e, em seguida, realiza-se a divisão. Logo: d = 3.333333.
  - Em (2), primeiro divide-se a por b e, em seguida, se faz a conversão explícita de tipo. Logo: d = 3.0.

Operadores de incremento e decremento

- Uma operação muito comum em programas de computador é **incrementar de 1** o valor da variável.

```
x = x + 1
```

- Como a operação incremento de 1 é muito comum, em C tem-se um operador especial: ++

- Ao invés de escrevermos **x = x + 1**, podemos escrever: **x++**

- Da mesma forma, para a operação decremento de 1: Para **x = x - 1**, podemos escrever: **x--**

Operações combinadas com a atribuição

- As operações de incremento (++) e decremento (--) são exemplos de operações combinadas com a atribuição.

- Na linguagem C, sempre que for necessário escrever uma operação de atribuição da forma:

```
variavel = variavel operador expressao;
```

poderemos combinar as operações.

Exemplos:

```
x = x + 5;          x += 5;
x = x - (a + b);    x -= (a + b);
x = x * (a - b);    x *= (a - b);
x = x / (x + 1);    x /= (x + 1);
```

Beecrowd

Criar conta:

- https://www.beecrowd.com.br/judge/pt/login
- **Usar o email institucional**

Acessar área da disciplina:

- https://www.beecrowd.com.br/judge/pt/disciplines/join
- **id: 14280**
- **key: `L-EUWPF`**

Referências

- Senne, E.L.F. Primeiro Curso de Programação em C. Editora: Visual Books; edição: 3; ano:2009; número de páginas: 318. Disponível na Biblioteca.

- BACKES, André. **Linguagem C:** completa e descomplicada. Rio de Janeiro: Elsevier, 2013. 371 p. ISBN 978-85-352-6855-3.Disponível na Biblioteca.