Estruturas básicas da linguagem C

Linguagem de alto nível

Definição

- A linguagem interna do computador é
  - conhecida como “linguagem de máquina”, e
  - muito complicada de usar na implementação de programas
- Uma linguagem de alto nível
  - está mais próxima da linguagem natural, e
  - é muito fácil de utilizar na implementação de programas

Tradutores

- Um programa numa linguagem de alto nível para poder ser "executado" (para funcionar) precisa de ser traduzido para linguagem de máquina
- Existem dois tipos de programas tradutores:
  - interpretadores
  - compiladores

Interpretador

- As intruções de um programa escrito em linguagem de alto nível (código-fonte) são traduzidas e executadas uma e uma,
  - cada vez que o programa é executado é necessário fazer a tradução,
    pois não é criada uma versão executável (ficheiro em liguagem de máquina)
- Exemplos de linguagens que usam interpretadores:
  - LISP
  - MatLab
  - PHP
  - JavaScript

Compilador

- Todas as instruções de um programa escrito em linguagem de alto nível são traduzidas de uma só vez
  - ou seja, o compilador cria uma versão executável do programa, bastando compilar o código-fonte uma única vez
- Exemplos de linguagens que usam compiladores:
  - C
  - Pascal
  - Java
  - C++
  - C#
- Os compiladores analisam o código-fonte em três fases:
  - análise léxica
  - análise sintática
  - análise semântica

Compilador

- Análise léxica
  - separação do código-fonte (texto) em palavras (os elementos/tokens léxicos)
    - lê o texto caráter a caráter, formando palavras (tokens) e identificando cada uma delas segundo as suas regras de construção (sintaxe)
    - se as regras de alguns tokens são violadas, são identificados **erros léxicos**
  - despreza os espaços em branco, as formatações e os comentários
- Análise sintática
  - verificação sintática/gramatical de construção das frases (cadeias de tokens) que compõem um texto, em que as frases são as **instruções** e o texto é o **programa**
    - verifica se a estrutura das frases seguem as regras da gramática formal da linguagem de programação (cada frase tem as suas regras de construção)
    - ou seja, verifica se existem **erros sintáticos** na sintaxe do programa
- Análise semântica
  - verificação de outras regras, associadas ao significado de cada instrução e do programa

Compilador

- Compilação e execução de um programa em linguagem C

- Exemplo (em UNIX e LINUX):
  - fazer um programa de nome **prog1.c** (usar o editor jpico)
  - compilar este programa:

    $ **gcc** –o **prog1 prog1.c**

    não havendo erros de qualquer tipo, o programa gcc cria o programa executável de nome **prog1**
  - para executar o programa, basta fazer:

    $ **./prog1**

Entidades da linguagem C

Alfbeto em C

- O conjunto básico de caracteres gráficos da linguagem C é o que consta na tabela que se segue

Forma	Membros
letra	A B C D E F G H I J K L M N O P Q R S T U V W X Y Z a b c d e f g h i j k l m n o p q r s t u v w x y z
dígito	0 1 2 3 4 5 6 7 8 9
underscore	_
pontuação	! " # % & ' ( ) * + , - . / : ; < = > [ ] \ ^ { } | ~

Palavras reservadas

- O conjunto básico de caracteres gráficos da linguagem C é o seguinte:

auto	double	**int**	**struct**
break	**else**	long	switch
case	enum	register	**typedef**
char	extern	**return**	union
const	**float**	short	unsigned
continue	**for**	signed	**void**
default	goto	**sizeof**	volatile
do	**if**	static	**while**

- Não podem ser usadas a não ser para aqueles fins pré-definidos
  - por exemplo, não podem ser usadas como nomes de variáveis
- A tentativa de utilização para outros fins resulta numa série de erros de compilação

Funções matemáticas predefinidas

- Biblioteca “math.h” (algumas funções)

Função	Descrição
sin(x)	seno de um valor real x (em radianos)
cos(x)	coseno de um valor real x (em radianos)
tan(x)	tangente de um valor real x (em radianos)
asin(x)	arco cujo seno é o valor real x (em radianos)
acos(x)	arco cujo coseno é o valor real x (em radianos)
atan(x)	arco cuja tangente é o valor real x (em radianos)
exp(x)	exponencial de um valor real x (e^x)
log(x)	logaritmo de um valor real positivo não nulo x (x > 0) na base **e**
log10(x)	logaritmo de um valor real positivo não nulo x (x > 0) na base **10**
sqrt(x)	raiz quadrada de um valor real positivo ou nulo (≥ 0) x
pow(x,y)	x elevado à potência y (x^y), x e y valores reais reais
floor(x)	maior inteiro que é menor ou igual ao real x
ceil(x)	menor inteiro que é maior ou igual ao real x,

Funções matemáticas predefinidas

- Embora não sejam palavras reservadas, não se devem usar para outros fins, pois isso pode gerar conflitos de nomes
- A tentativa de utilização para outros fins pode ter como consequência uma série de erros de compilação

Nomes ou identificadores

- Há 3 classes de nomes ou identificadores:

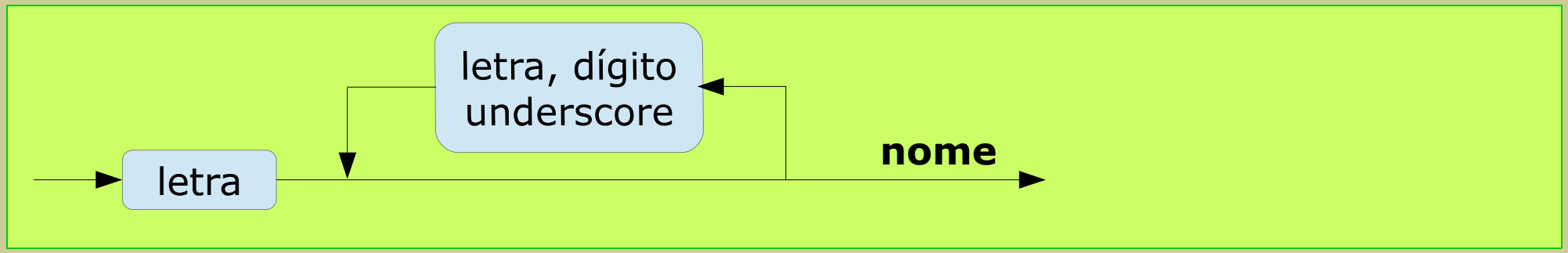
Obedecem a uma sintaxe

Podem ser redefinidos,
mas não é conveniente

Não podem ser redefinidos

Nomes ou identificadores

- As regras de formação de nomes/identificadores são as seguintes
  - **diferenciação gráfica** (maiúsculas e minúsculas)
  - **tamanho variável** (ANSI C limitado a 31 caracteres)
  - **validade** (caráter inicial é: a...z, A...Z)
    - **nomes válidos**: j, j5, mat3
    - **nomes inválidos**: 5j, j5%, const
- Um nome ou identificador,
  - deve começar sempre com uma letra,
  - pode ser seguida por letras, digitos e alguns outros carateres (underscore: _)
- Sintaxe (diagrama)



Estrutura de um programa em C

Sintaxe

Diretivas para o compilador (do pré-processador)
void/int main ()
{
  Declarações
  Instruções
}

Comentários

- Não são traduzidos pelo compilador em instruções de linguagem de máquina
- Servem apenas para tornar o código-fonte mais "claro"
- Sintaxe:

  /* comentários ***/***

  // comentários

Diretivas para o compilador (do pré-processador)

- Permite que o programador modifique a compilação
- O pré-processador é um programa que examina e modifica o código-fonte antes da compilação
- As diretivas são os comandos utilizados pelo pré-processador
- Estes comandos estão disponíveis no código-fonte, mas não no código compilado
- As diretivas iniciam-se com **#**

A diretiva #include

- Principal diretiva, pois é usado em praticamente todos os programas em C
- Permite inserir um arquivo (biblioteca de funções) qualquer no código-fonte
- A diretiva **include** é substituída pelo conteúdo do arquivo
- Usa-se **<>** para indicar que o arquivo é procurado apenas na pasta *include* (pasta criada automaticamente aquando da instalação do compilador)
- Usa-se **“”** para indicar que o arquivo é procurado na pasta atual, e se não for encontrado é procurado na pasta *include*
- As bibliotecas mais usadas são:

  #include <stdio.h>

  - usada em praticamente todos os programas
  - inclui as funções de entrada e saída (*input* e *output*) de dados: **scanf, printf, ...**

  #include <math.h>

  - inclui as funções matemáticas: **sqrt, pow, sin, cos, tan, log, ...**

Diretiva #define

- Permite definir constantes sem consumir memória durante a execução
- Exemplos:

```
#define  PI  3.141592
#define  ERRO  printf("Erro na introducao dos dados.\n");
```

- Permite definir trechos de código com parâmetros (macros)
  - não pode ter espaços no identificador (ex: soma (x,y))
  - exemplos:

```
#define  soma(x,y)  (x) + (y)
#define  max(a,b)   ((a) > (b) ? (a) : (b))
```

main

- A função principal (programa propriamente dito)
- Formado por:
  - Declarações
  - Instruções
- As *Declarações* devem aparecer antes das *Instruções*
- As declarações mais comuns são das variáveis que o programa pode utilizar (indicadas nas *Instruções*)
- Na declaração de uma variável, o nome desta segue a sintaxe de nomes ou identificadores (ver antes)
- As declarações e as instruções terminam sempre com ponto e vírgula (**;**)
- A execução do programa começa na primeira instrução e segue até à última, sequencialmente

main

- Exemplo:

```
#include <stdio.h>
void main ()
{
  int  a, b, soma;
  printf("Introduza o primeiro valor:\n");
  scanf("%d", &a);
  printf("Introduza o segundo valor:\n");
  scanf("%d", &b);
  soma = a + b;
  printf("A soma é %d.\n", soma);
}
```

main

- **Exemplo:**

```
#include <stdio.h>                                   diretiva
...
        int   a, b, soma;                            declaração
...
    printf("Introduza o primeiro valor:\n");
    scanf("%d", &a);
    printf("Introduza o segundo valor:\n");          instruções
    scanf("%d", &b);
    soma = a + b;
    printf("A soma é %d.\n", soma);
...
```

Tipos de Dados Simples

A linguagem C suporta 3 tipos de dados simples:

- Inteiros
- Reais
- Carateres

Inteiros

- A representação de um número inteiro, em formato decimal, é composta por duas partes (pela ordem apresentada):
  - um sinal (+, -), sendo que o sinal + (positivo) pode ser omitido,
  - uma sequência de dígitos
- Exemplos: 2563, -753, +35

Inteiros - operadores aritméticos

+ soma

Ex: 10 **+** 3 (= 13)

– subtração

Ex: 12 **–** 2 (= 10)

***** multiplicação/produto

Ex: 5 ***** 10 (= 50)

/ divisão (inteira)

Ex: 20 **/** 3 (= 6)

% resto da divisão (inteira)

Ex: 20 **%** 3 (= 2)

Inteiros - declaração de variáveis

- Sintaxe para declarar variáveis do tipo inteiro (identificar todas as variáveis)
  - Sintaxe para declarar apenas uma variável do tipo inteiro

```
int  variável;     (; obrigatório no fim da declaração)
```

  - Sintaxe para declarar duas variáveis do tipo inteiro

```
int  variável1, variável2;     (; obrigatório no fim da declaração)
```

  - ...
- Exemplos:

```
int  a;
int  i, j;
int  ab, produto, pt_3, soma, k;
```

- Uma variável do tipo inteiro ocupa 4 bytes (32 bits) na memória; assim,
  - maior inteiro: 2 147 483 647
  - menor inteiro: -2 147 483 648

Reais

- Um número real pode ser representado usando dois formatos:
  - com ponto fixo (ex: 12.34)
  - com ponto flutuante (ex: $0.4273x10^{-8}$)
- Na representação com ponto fixo, um número é composta por 2 partes:
  - uma parte inteira (número inteiro com ou sem sinal)
  - uma parte fracionária (ponto decimal seguido por inteiro sem sinal)
- Na representação em formato com ponto flutuante, a notação é semelhante à notação científica, que é composta por 3 partes:
  - um número real em formato de ponto fixo ou um inteiro, com ou sem sinal,
  - o caráter 'e' ou 'E'
  - um número inteiro com ou sem sinal
- Exemplos:
  - formato com ponto fixo: 12.764 (12 e 0.764), -542.78 (-542 e 0.78)
  - com expoente: 12.78E-6, -34E10

Reais - operadores aritméticos

+ soma

Ex: 1.2 **+** 3.0 (= 4.2)

– subtração

Ex: 4.12 **–** 2.3 (= 1.82)

***** multiplicação/produto

Ex: 5.2 ***** 10.0 (= 52.0)

/ divisão (real)

Ex: 10.0 **/** 4.0 (= 2.5)

Reais - declaração de variáveis

- Sintaxe para declarar variáveis do tipo real (identiicar todas as variáveis)
  - Sintaxe para declarar apenas uma variável do tipo real

float variável**;** (**;** obrigatório no fim da declaração)

  - Sintaxe para declarar duas variáveis do tipo real

float variável1, variável2**;** (**;** obrigatório no fim da declaração)

  - ...
- Exemplos:

float x;

float X, xyz, ponto, soma;

- Uma variável do tipo real ocupa 4 bytes (32 bits) na memória; assim,
  - maior número real (valor absoluto): 3.40×10^{38}
  - menor número real (valor absoluto): 1.18×10^{-38}

Carateres

- Um caráter é representado entre plicas (')
- Exemplos:
  - 'a'
  - 'A'
  - 'x'
  - 'n'
- Note-se que existe diferença entre letras maiúsculas e minúsculas:

  'a' e 'A' são diferentes (carateres diferentes)
- Alguns carateres especiais:
  - \a (sinal sonoro - *bell*)
  - \b (recua uma posição – *backspace*)
  - \n (nova linha – *new line*)
  - \t (tabulação - *tab*)

Carateres - declaração de variáveis

- Sintaxe para declarar variáveis do tipo caráter (identificar todas as variáveis)
  - Sintaxe para declarar apenas uma variável do tipo caráter

```
char  variável1;     (; obrigatório no fim da declaração)
```

  - Sintaxe para declarar duas variáveis do tipo caráter

```
char  variável1, variável2;   (; obrigatório no fim da declaração)
```

  - ...
- Exemplos:

```
char  c;
char  a, ch;
char  resposta, opcao, marca, p;
```

- Um caráter ocupa 1 byte (8 bits) na memória

Expressões aritméticas

- Uma expressão aritmética contém
  - operadores aritméticos (+, -, *, /, %)
  - valores numéricos (inteiros e reais)
  - variáveis do tipo numérico com valores
  - funções matemáticas predefinidas
- O computador calcula o valor de uma expressão segundo as seguintes regras:
  - se contiver apenas valores inteiros o resultado é um valor inteiro
  - se contiver pelo menos um valor real, o resultado é um valor real
  - os operadores (*, / e %) tem maior prioridade do que (+, -)
  - os cálculos são efetuados da esquerda para à direita, quando os operadores têm a mesma prioridade
  - as operações que estão entre parênteses são efetuadas em primeiro

Expressões aritméticas – exemplos

2 / 4	(= 0)
2.0 / 4	(= 0.5)
2 + 3 * 4	(= 14)
(2 + 3) * 4	(= 20)
8 / 4 / 2	(= 1)
8 / (4 / 2)	(= 4)

Operadores relacionais

- Os operadores relacionais permitam
  - comparar valores (ou expressões) dos tipos:
    - inteiro,
    - real, ou
    - caráter
  - devolvendo como resultado:
    - *verdadeiro* (*true*)
    - *falso* (*false*)
- Os operadores relacionais têm **menos prioridade** do que os aritméticos
- Os valores *verdadeiro* e *falso* são chamados valores lógicos, que não existem na linguagem C:
  - o valor **1** equivale ao *verdadeiro*
  - o valor **0** equivale ao *falso*

Operadores relacionais - símbolos

==	igual
>	maior
>=	maior ou igual
<	menor
<=	menor ou igual
!=	diferente

Operadores relacionais - exemplos

```
int  a, b, c, d, e;
a = 1; b = 2; c = 4; d = 5; e = -1;
```

a == b - 1	*verdadeiro*
c == d - e	*falso*
a == (b + 1) / 2	*verdadeiro*

Operadores lógicos

- Os operadores lógicos
  - permitem comparar expressões que tenham valores lógicos,
  - devolvendo também um valor lógico: verdadeiro (true) ou falso (false)
- Os operadores lógicos são os operadores com **menos prioridade** entre todos os operadores

Operadores lógicos - símbolos

&&	e (*and*)
||	ou (*or*)
!	não (*not*)

Operadores lógicos - exemplos

```
int  a, b, c, d, e;
a = 5; b = 4; c = 9; d = -2; e = 3;
```

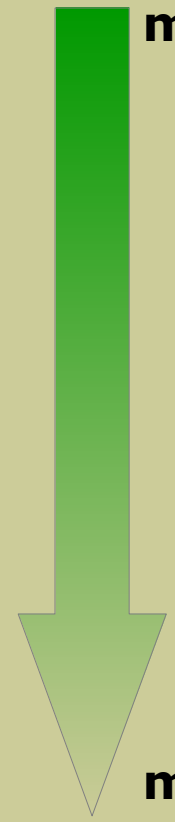
!(2 == 3) && 1 == 1	*verdadeiro*
a + b == c || d >= e	*verdadeiro*
a == d - e && e <= b	*falso*
!(a + 1 == b) || a + 1 == b	*verdadeiro*

Expressões lógicas

- Uma expressão lógica contém
  - operadores relacionais (==, !=, >, >=, <, <=)
  - operadores lógicos (&&, ||, !)
  - expressões matemáticas
  - carateres

Prioridade dos operadores

- Operadores unários: **! + −**
- Operadores aritméticos: *** / %**
- Operadores aritméticos: **+ −**
- Operadores relacionais: **< <= > >=**
- Operadores relacionais: **== !=**
- Operadores lógicos: **&&**
- Operadores lógicos: **||**



maior prioridade

menor prioridade