3. Linguagem de Programação C

Prof. Renato Tinós

Departamento de Computação e Matemática
(FFCLRP/USP)

Principais Tópicos

3.2. Estrutura de Programas e Representação de Dados

3.2.1. Estrutura de programas em C

3.2.2. Representação de Dados

3.2.2.1. Constantes

3.2.2.2. Variáveis

3.2.2.3. Tipos de Dados

3.2.1. Estrutura de Programas em C

- **Exemplo de um programa em C**
  - Seja um programa em C que gera uma tabela comparando os valores de N^2 e 2^N

N	N^2	2^N
0	0	1
1	1	2
2	4	4
3	9	8
4	16	16
5	25	32

3.2.1. Estrutura de Programas em C

```
/* arquivo: potencia.c
    este programa gera uma tabela comparando
    valores das funcoes n^2 e 2^n                */

#include <stdio.h>
#include <genlib.h>

/* Constantes
* LimInf - Limite inferior
* LimSup - Limite superior */
#define LimInf 0
#define LimSup 12
```

Comentários do programa

Inclusão de bibliotecas

Seção de comentários

Definições de constantes

3.2.1. Estrutura de Programas em C

```
/* programa potencia
* Uso: potencia (n, k) retorna n elevado a k */
int potencia (int n, int k){
     int i, resultado;

    resultado = 1;
    for (i = 0; i < k; i++){
          resultado*= n;
    }
    return (resultado);
}

// Programa principal
main (){
     int j;

     printf (“      |     2 |     N  \n”);
     printf (“  N |   N  |    2 \n”);
     printf (“----+-----+------- \n”);
     for (j = LimInf; j <= LimSup; j++){
          printf (“ %2d  |  %3d  |  %4d \n”, j, potencia (j, 2), potencia (2, j));
     }
}
```

Comentários de função

Função

Definição de variáveis locais

Programa Principal

3.2.1. Estrutura de Programas em C

- **Comentários**
  - Ignorados pelo compilador,
  - Fornecem informações importantes
    - » Para o programador
    - » Para outros programadores
  - Formatos:

```
// Comentário de uma linha

/* Comentário de
   várias linhas*/
```

3.2.1. Estrutura de Programas em C

- **Bibliotecas**
  - Conjunto de ferramentas escritas anteriormente que executam operações úteis
    - » Ao serem incluídas, suas ferramentas são disponibilizadas

#include <stdio.h>

- » Biblioteca do sistema, fornecida com o ANSI C
- » Contém funções de entrada e saída

#include <genlib.h>

- » Biblioteca privada

3.2.1. Estrutura de Programas em C

- **Funções**
  - Programas grandes são difíceis de entender
    - » Maioria dos programas são divididos em programas menores (funções)
  - Grande parte de processamento em um programa C é geralmente realizado por meio de funções
  - Função é uma unidade de código que:
    - » Realiza uma operação específica
    - » É identificada por um nome

3.2.1. Estrutura de Programas em C

- **Função**
  - No programa *potencia.c* existem duas funções
    - » *int potencia (int n, int k)*
    - » *main ( )*
  - Funções de C retornam valores através do comando *return*
    - » Ex.: *return (x);*
      - x é uma variável

3.2.1. Estrutura de Programas em C

- **Programa principal**
  - Todo programa C contém uma função chamada *main*
    - » Especifica o ponto inicial para execução do programa
    - » Chamada quando o programa começa sua execução
      - Quando *main* termina sua execução, o programa termina
    - » Pode retornar um valor do tipo *int*

3.2.2.1. Constantes

- **Declaração de constantes**
  - São globais
    - » Válidas em qualquer linha do programa
  - Programas podem incluir definições que se aplicam a todo o programa (constantes)

    #define LimInf 0

    #define LimSup 12
  - Formato: **#define** *nome valor*
    - » Define *nome* como equivalente a *valor*

3.2.2.1. Constantes

- **Vantagens em associar nomes simbólicos a constantes**
  - O leitor do programa entende mais facilmente o significado das constantes
  - A centralização de todas as definições no início do arquivo facilita a alteração do valor associado ao nome
    - » Ex.: para alterar os limites do programa ***potencia.c*** é necessário apenas mudar os valores das constantes

3.2.2.2. Variáveis

- **Uma das principais características dos programas é que eles manipulam dados**
  - Programas precisam armazenar dados
  - Programas trabalham com vários tipos de dados diferentes
    - » Números
    - » Texto
    - » Estruturas de dados mais sofisticadas
  - Geralmente, dados são armazenados em variáveis

3.2.2.2. Variáveis

```
/* Programa: Apresenta o valor de uma soma simples*/
# include <stdio.h>

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

    a = 7;
    b = 2;
    valor = a + b;                                    // soma
    printf ("Valor da expressao: %d \n", valor);
}
```

3.2.2.2. Variáveis

```
/* Programa: imprime operacoes basicas sobre dois numeros */
# include <stdio.h>

main () {
    int soma, div, sub, mult, resto;
    int a, b;

    a = 7;
    b = 2;
    soma = a + b;                                   // soma
    sub = a - b;                                    // subtracao
    mult = a* b;                                    // multiplicacao
    div = a / b;                                    // divisao
    resto = a % b;                                  // resto
    printf ("Soma = %d, subtracao = %d,
                multiplicacao = %d, divisao o = %d,
                resto = %d \n", soma, sub, mult, div, resto);
}
```

3.2.2.2. Variáveis

Exercício 3.2.1. Escreva e implemente um programa em C que imprima a soma e a média aritmética dos números 56, 109 e 345 .

3.2.2.2. Variáveis

- **Toda variável utilizada no programa deve ser declarada**
- **Sintaxe para a declaração de variáveis em C**

```
tipo lista_de_nomes;
```

- **Em C, o valor inicial de uma variável é indefinido**
  - Exceção:

```
int result = 0;
```

- **Valor inicial de uma declaração é chamado de inicializador**

3.2.2.2. Variáveis

- **Uma variável tem as seguintes propriedades:**
  - Nome
  - Tipo
  - Tempo de vida
  - Escopo (área de atuação)
    - » Variáveis locais
    - » Variáveis globais

3.2.2.2. Variáveis

- **Convenções para nomes**

  - Identificadores: nomes usados para variáveis, constantes, tipos e funções
  - Regras
    - » Nome deve iniciar com letra ou *underscore* (_)
    - » Caracteres de um nome devem ser letras, números ou *underscores*
    - » Palavras chaves não podem ser utilizadas como nomes

3.2.2.2. Variáveis

- **Palavras chaves**

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*

3.2.2.2. Variáveis

- **Convenções para nomes**
  - Letras maiúsculas e minúsculas são consideradas diferentes
  - Estilo de programação pode ser melhorado adotando convenções
    - » Exemplos que podem ser adotados (não obrigatório):
      - Nomes de variáveis e tipos de dados começam com letras minúsculas
      - Nomes de funções e constantes começam com letras MAIÚSCULAS

3.2.2.2. Variáveis

- **Variáveis locais**
  - Declaradas no início de uma função
  - Escopo é a função onde ela aparece
    - » Outras funções não têm acesso direto a elas
      - Exceto as funções internas
  - Tempo de vida = enquanto a função estiver ativa
    - » Chamada da função aloca espaço para as variáveis
    - » Término de execução da função libera o espaço alocado

3.2.2.2. Variáveis

- **Variáveis globais**
  - Aparecem fora de qualquer definição de função
  - Escopo é o resto do arquivo onde é declarada
  - Tempo de vida é o tempo de execução do programa
  - Não devem ser utilizadas em excesso
  - Permite que funções possam interferir umas com as outras

3.2.2.3. Tipos de Dados

- **A linguagem C requer que todas as variáveis sejam declaradas para restringir seus conteúdos a valores de um tipo particular**
- **Tipos de dados são definidos por duas propriedades:**
  - Domínio
    - » Conjunto de valores que pertencem ao tipo
  - Conjunto de operações
    - » Operações que podem ser realizadas sobre valores do tipo
    - » Define o comportamento do tipo

3.2.2.3. Tipos de Dados

- **Exemplo:**
  - O domínio do tipo *int* inclui todos os inteiros (... -2,-1,0,1,2 ...) permitidos pelo hardware do computador
  - Conjunto de operações inclui, por exemplo, o conjunto das operações aritméticas padrão (adição, subtração, multiplicação e divisão)
  - Outros tipos podem ter domínio e conjunto de operações diferentes

3.2.2.3. Tipos de Dados

- **Muito do poder de linguagens de alto nível vem da possibilidade de definir novos tipos a partir dos tipos existentes**

- **ANSI possui um conjunto de tipos fundamentais (tipos atômicos)**
  - Ex.: *integer*, *floating-point* e *char*

3.2.2.3. Tipos de Dados

- **Tipo inteiro**
  - valores inteiros (...,-1,0,1,2,3,...)
  - faixa de valores pode variar. Exemplos:
    - » 16 bits
      - comporta 65.536 valores (2^{16})
    - » 32 bits
      - comporta 4.294.962.2.296 valores (2^{32})

3.2.2.3. Tipos de Dados

Type	Explanation	Format Specifier
char	Smallest addressable unit of the machine that can contain basic character set. It is an integer type. Actual type can be either signed or unsigned. It contains `CHAR_BIT` bits.[3]	%c
signed char	Of the same size as `char`, but guaranteed to be signed. Capable of containing **at least** the [−127, +127] range;[3][4]	%c (or %hhi for numerical output)
unsigned char	Of the same size as `char`, but guaranteed to be unsigned. Contains **at least** the [0, 255] range.[5]	%c (or %hhu for numerical output)
short short int signed short signed short int	*Short* signed integer type. Capable of containing **at least** the [−32,767, +32,767] range;[3][4] thus, it is at least 16 bits in size. The negative value is −32767 (not −32768) due to the one's-complement and sign-magnitude representations allowed by the standard, though the two's-complement representation is much more common.[6]	%hi
unsigned short unsigned short int	*Short* unsigned integer type. Contains **at least** the [0, 65535] range;[3][4]	%hu
int signed signed int	Basic signed integer type. Capable of containing **at least** the [−32,767, +32,767] range;[3][4] thus, it is at least 16 bits in size.	%i or %d
unsigned unsigned int	Basic unsigned integer type. Contains **at least** the [0, 65535] range;[3][4]	%u
long long int signed long signed long int	*Long* signed integer type. Capable of containing **at least** the [−2,147,483,647, +2,147,483,647] range;[3][4] thus, it is at least 32 bits in size.	%li
unsigned long unsigned long int	*Long* unsigned integer type. Capable of containing **at least** the [0, 4,294,967,295] range;[3][4]	%lu
long long long long int signed long long signed long long int	*Long long* signed integer type. Capable of containing **at least** the [−9,223,372,036,854,775,807, +9,223,372,036,854,775,807] range;[3][4] thus, it is at least 64 bits in size. Specified since the C99 version of the standard.	%lli
unsigned long long unsigned long long int	*Long long* unsigned integer type. Contains **at least** the [0, +18,446,744,073,709,551,615] range;[3][4] Specified since the C99 version of the standard.	%llu

3.2.2.3. Tipos de Dados

Properties

The following table summarizes all available integer types and their properties:

Type specifier	Equivalent type	Width in bits by data model				
		C++ standard	LP32	ILP32	LLP64	LP64
short	short int	at least 16	16	16	16	16
short int						
signed short						
signed short int						
unsigned short	unsigned short int					
unsigned short int						
int	int	at least 16	16	32	32	32
signed						
signed int						
unsigned	unsigned int					
unsigned int						
long	long int	at least 32	32	32	32	64
long int						
signed long						
signed long int						
unsigned long	unsigned long int					
unsigned long int						
long long	long long int (C++11)	at least 64	64	64	64	64
long long int						
signed long long						
signed long long int						
unsigned long long	unsigned long long int (C++11)					
unsigned long long int						

Note: the C++ Standard guarantees that
`1 == sizeof(char) <= sizeof(short) <= sizeof(int) <= sizeof(long) <= sizeof(long long).`

Data models

32 bit systems:

- **LP32**:
  - Win16 API
- **ILP32**:
  - Win32 API
  - Unix and Unix-like systems (Linux, Mac OS X)

64 bit systems:

- **LLP64**
  - Win64 API
- **LP64**
  - Unix and Unix-like systems (Linux, Mac OS X)

3.2.2.3. Tipos de Dados

- **Tipo inteiro**
  - Bases numéricas
    - » Valores inteiros geralmente são definidos na base decimal
    - » Base hexadecimal: começar o número com 0x
      - Exemplo: 0xFF

3.2.2.3. Tipos de Dados

- **Tipo ponto flutuante**
  - Representam números reais
  - Conseguem representar uma grande quantidade de números
    - » Uso da notação científica
  - *float*
    - » 32 bits
    - » Representatividade: $\pm$ 3,4 x 10^{-38} e $\pm$ 3,4 x 10^{+38}
    - » 6 a 7 dígitos de precisão
  - *double*
    - » 64 bits
    - » Representatividade: $\pm$ 1,7 x 10^{-308} e $\pm$ 1,7 x 10^{+308}
    - » 14 a 15 dígitos de precisão
    - » *long double*
      - 80 bits

3.2.2.3. Tipos de Dados

float	Real floating-point type, usually referred to as a single-precision floating-point type. Actual properties unspecified (except minimum limits), however on most systems this is the IEEE 754 single-precision binary floating-point format (32 bits). This format is required by the optional Annex F "IEC 60559 floating-point arithmetic".	for formatted input: %f %F for digital notation, or %g %G, or %e %E %a %A for scientific notation[7]
double	Real floating-point type, usually referred to as a double-precision floating-point type. Actual properties unspecified (except minimum limits), however on most systems this is the IEEE 754 double-precision binary floating-point format (64 bits). This format is required by the optional Annex F "IEC 60559 floating-point arithmetic".	%lf %lF %lg %lG %le %lE %la %lA;[7] for formatted output, the length modifier l is optional.
long double	Real floating-point type, usually mapped to an extended precision floating-point number format. Actual properties unspecified. It can be either x86 extended-precision floating-point format (80 bits, but typically 96 bits or 128 bits in memory with padding bytes), the non-IEEE "double-double" (128 bits), IEEE 754 quadruple-precision floating-point format (128 bits), or the same as double. See the article on long double for details.	%Lf %LF %Lg %LG %Le %LE %La %LA[7]

3.2.2.3. Tipos de Dados

- **Tipos caractere (*char*)**

  – Símbolos individuais que podem aparecer no vídeo ao ser digitados no teclado

    » Letras, dígitos, marcas de pontuação, barra de espaço, *return*, etc.

    » Tamanho típico: 1 byte

    » Internamente, estes valores são representados associando cada caractere a um código ASCII

    » Tradução é automática

3.2.2.3. Tipos de Dados

TABLE 1-1 ASCII codes

	0	1	2	3	4	5	6	7	8	9
0	\000	\001	\002	\003	\004	\005	\006	\a	\b	\t
10	\n	\v	\f	\r	\016	\017	\020	\021	\022	\023
20	\024	\025	\026	\027	\030	\031	\032	\033	\034	\035
30	\036	\037	*space*	!	"	#	$	%	&	'
40	(	)	*	+	,	-	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	[	\	]	^	_	`	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	{	|	}	~	\177		

3.2.2.3. Tipos de Dados

- **Caracteres em C**
  - Caracteres padrão
    - » ‘A’, ‘b’, ‘8’
  - Caracteres especiais (sequências de escape)
    - » Sequências de dois caracteres (começando com o caractere \ )
  - Sequência de caracteres = “*string*”

3.2.2.3. Tipos de Dados

TABLE 1-2 Escape sequences

`'\a'`	The alert character (the terminal beeps)
`'\b'`	Backspace
`'\f'`	Formfeed (starts a new page)
`'\n'`	Newline
`'\r'`	Return (returns to the beginning of the line without advancing)
`'\t'`	Tab
`'\v'`	Vertical tab
`'\\'`	The character \ itself
`'\''`	The character ' (the backslash is required only in single characters)
`'\"'`	The character " (the backslash is required only in strings)
`'\`*ddd*`'`	The character whose ASCII code is the octal (base 8) number *ddd*
`'\x`*dd*`'`	The character whose ASCII code is the hex (base 16) number *dd*
`'\0'`	The null character (with zero as its character code)