SCC-120

INTRODUÇÃO À CIÊNCIA DA COMPUTAÇÃO

Prof. Zhao Liang

O QUE É C ?

- No inicial do computador, os programas eram escritos em linguagem de maquina
  - Instruções primitivas que podiam ser executadas diretamente na maquina

    Ex.: *mov* A B
  - Estrutura da linguagem refletia da estrutura da maquina
    - Não reflete as necessidades do programador;
    - Difícil de programar

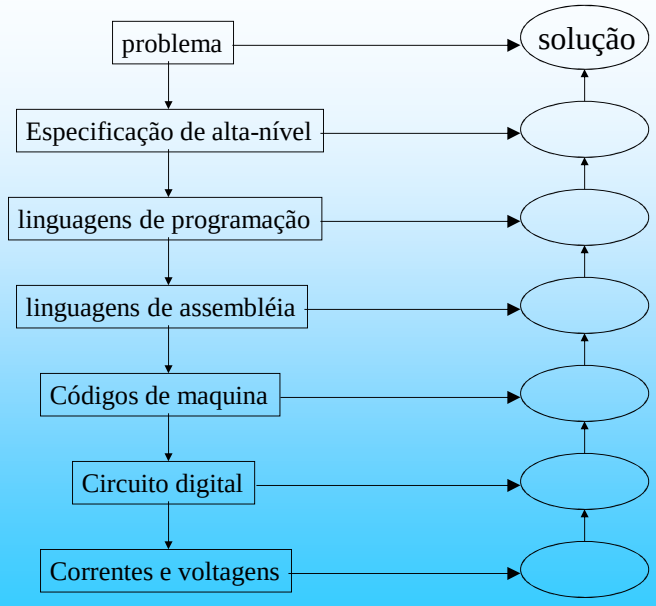
O QUE É C ?

- Maior parte da computação envolvia o calculo de formulas
  - formulas eram traduzidas para linguagem de maquinas
  - Por que não escrever programas parecidos com as formulas que se deseja computar?
- Em 1950, grupo de programadores da IBM liderados por John Backus produz a versão inicial da linguagem FORTRAN (formula transform);
  - Primeira linguagem de alto nível;

O QUE É C ?

- Varias outras linguagens de alta nível foram criadas:
  - Uma das mais bem sucedidas foi uma linguagem chamada C;
  - Criada em 1972 nos laboratórios por Denis Richie;
  - Revisada e padronizada polo ANSI em 1989;

O QUE É C ?

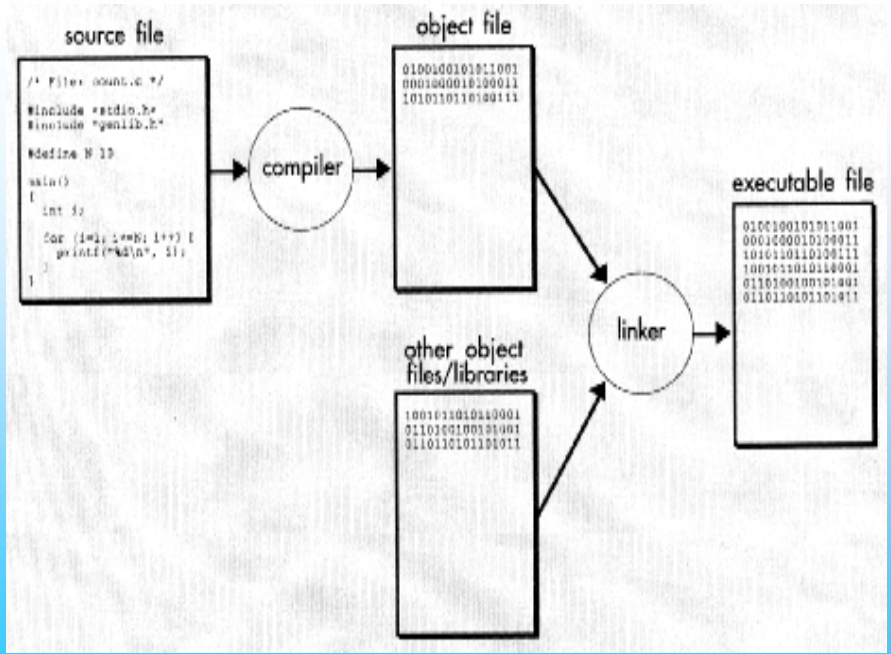


O QUE É C ?

- Uma linguagem de programação de alto nível;
- Uma linguagem de programação de propósito geral;
- Uma das linguagens mais utilizadas atualmente;
- Existem vários compiladores C.

Escrevendo um programa em C

- Criar arquivo fonte (arquivo que contem o texto do programa);
- Traduzir arquivo fonte para um arquivo executável
  - ♦ Compilador transforma arquivo fonte em um arquivo objeto (contem as instruções correspondentes em linguagem de maq.);
  - ♦ Arquivo objeto é combinado com outros arquivos objeto para produzir arquivo executável;
    - ♦ Outros arquivos objetos são arquivos pré-definidos (libraries); Contem instruções em Ling. de maq. Para varias operações geralmente utilizadas pelos programas;
    - ♦ O processo de combinação dos arquivos objetos em um arquivo executável é chamado de ligação (linking);



Estrutura de um programa C

Exemplo de um programa em C

- Seja um programa em C que gera os valores de N^2 e 2^N

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

```
/* File: powertab.c
   this program generates a table comparing
   values of the functions N^2 and 2^N.
*/
#include <stdio.h>
#include "genlib.h"
/* constantes
   LowerLimit - starting value for the table
   UpperLimit - final value for the table
*/
#define LowerLimit = 0
#define UpperLimit = 12

/* Privite function prototypes */
static int RaiseIntToPower (int n, int k);
```

```
/* main program */
main ( )
{
    int n;

    printf(“   N    |    N^2  |   2^N   \n”);
    for(n = LowerLimit; n <= UpperLimit; n++)
            printf(“  %2d       %3d        %4d    \n”,   n,
                    RaiseIntToPower(2, n),
                    RaiseIntToPower(n, 2));
}
```

```
/* Function: RaiseIntToPower
   Usage: p = RaiseIntToPower(n, k);
   ----------------------------------------
   This function returns the n to the kth power */
static int RaiseIntToPower(int n, int k)
{
    int i, result;

    result = 1;
    for(i = 0; i <= k; i++)
        result *= n;
    return (result);
}
```

A Forma Geral de um Programa em C

```
declarações globais
protótipos de funções
tipo devolvido  main (lista de parâmetros)
{
     seqüência de comandos
}
tipo devolvido f1(lista de parâmetros)
{
     seqüência de comandos
}
...
tipo devolvido fN(lista de parâmetros)
{
     seqüência de comandos
}
```

Variáveis

- Uma variável é uma posição nomeada de memória, que é usada para guardar um valor que pode ser modificado pelo programa;
- Todas variáveis devem ser declaradas antes de serem usadas;
- A formato geral de uma declaração é

tipo lista_de_variáveis;

Variáveis

- Exemplos:

```
int  i, j, l;
short int  si;
unsigned int  ui;
double  balance, profit, loss;
int  x = 5;
char ch;
char str[100];
int vet[100][100];
```

Variáveis

- **Inicialização de Variáveis**

tipo nome-da_variável = constante;

- Exemplos:

```
char ch = ‘a’;
int first = 0;
float balance = 123.23;
```

- O valor inicial de uma declaração é chamado de *inicializador*.

Variáveis

- Uma variável tem as seguintes propriedades:
  - nome
  - tipo
  - tempo de vida
  - escopo
    - variáveis globais
    - variáveis locais

Variáveis

Convenções para Nomes

- Identificadores: nomes usados para variáveis, constantes, tipos e funções.
- Regras
  - nome deve iniciar com *letras* ou *underscore* (_);
  - caracteres de um nome devem ser letras, números ou underscores;
  - palavras chaves não podem ser usadas como nomes;
  - letras maiúsculas e minúsculas são consideradas diferentes

Uma lista de palavras-chave de C ANSI

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

Variáveis

- **Variáveis locais**
  - declaradas no inicial de uma função
  - escopo é a função onde ela aparece
  - outras funções não tem acesso direto a elas (menos as funções internas)
  - tempo de vida - enquanto a função estiver ativa
    - chamada da função aloca espaço para as variáveis;
    - termino da execução da função libera o espaço

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
  - Funções podem interferir umas com as outras

Tipos de Dados

- **Tipo Inteiro**
  - Vários tamanhos diferentes
    - **int** - 16 bits - valor máximo é 32767 (2^{15}-1)
    - **short** (int) - 16 bits, com a mesma margem
    - **long** (int) - 32 bits, valor máximo é 2.147.483.647 (2^{31}-1)
  - Palavra **unsigned** antes de cada tipo inteiro cria novos tipos
    - permite apenas valores não negativos;
    - **unsigned int** pode ser abreviado como **unsigned** (0 - 65535)
    - **unsigned long int** (0 - 4.294.967.295)

Tipos de Dados

- **Tipo Inteiro (cont.)**
  - Bases numéricas
    - Valores inteiros geralmente são definidos em base decimal;
    - Base octal - começa o numero com 0

      int oct = 040 /* 32 em decimal */
    - Base hexadecimal - começa o numero com 0x

      int hex = 0xFF /* 255 em hexadecimal */

Tipos de Dados

- **Tipos Ponto Flutuante**
  - Aumentam precisão as custas de mais espaço
  - Variantes
    - **float** - 32 bits
    - **double** - 64 bits
    - **long double** - 128 bits
  - Exemplos

    4.0, 2.34E+9

Tipos de Dados

- **Tipos Texto (char)**
  - ♦ Símbolos individuais que podem aparecer no vídeo ou ser digitados no teclado
    - letras, dígitos, marcas de pontuação, barra de espaço, return, etc.
    - Internamente estes valores são representados associando cada caracter a um código ASCII (American Standard Code for Information Interchange);
    - Tradução é automática

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		

Tipos de Dados

- **Tipo String**

  C não define explicitamente o tipo string, um string é representado por um vetor de caracteres.

- **Tipo Booleano (bool)**
  - utilizado para testar condições;
  - Assumem dois valores - *true* (1) e *false* (0)

Todos os tipos de dados definidos no padrao C ANSI

Tipo	Num de bits	Intervalo
char	8	-128 a 127
unsigned char	8	0 a 255
signed char	8	-128 a 127
int	16	-32.768 a 32.767
unsigned int	16	0 a 65.535
signed int	16	-32.768 a 32.767
short int	16	-32.768 a 32.767
unsigned short int	16	0 a 65.535
signed short int	16	-32.768 a 32.767
long int	32	-2.147.483.648 a 2.147.483.647
signed long int	32	-2.147.483.648 a 2.147.483.647
unsigned long int	32	0 a 4.294.967.295
float	32	3,4E-38 a 3.4E+38
double	64	1,7E-308 a 1,7E+308
long double	80	3,4E-4932 a 3,4E+4932

Operadores

- **Operador de Atribuição**

  nome_da_variável = expressão;

Ex.:

```
main( )
{
   int x = 5;
   int y;

   y = x + 3;
}
```

Operador de Atribuição

- **Conversão de tipos em Atribuições**
  - ♦ Regra de conversão de tipos:
    - O valor do lado direto é convertido no tipo do lado esquerdo;
    - Quando se converte de inteiros para caracteres, inteiros longo para inteiros e inteiros para inteiros curtos, a regra básica que a quantidade apropriada de bits mais significativos será ignorada.

Operador de Atribuição

Exemplo,

```
int x;
char ch;
float f;
void func(void)
{
   ch = x;          /* linha 1 - o valor de ch reflete apenas os 8
                       bits menos significativos; */
   x = f;           /* linha 2 - x recebe a parte inteira de f */
   f = ch;          /* linha 3 - f converte ch (8 bits) no mesmo
                       valor em formato ponto flutuante; */
   f = x;           /* linha 4 - f converte x (16 bits) no mesmo
                       valor em formato ponto flutuante */
}
```

Operador de Atribuição

- **Atribuições Múltiplas**

  C permite que você atribua o mesmo valor a muitas variáveis usando atribuições múltiplas em um único comando.

  Ex.: x = y = z = 0;

Operadores Aritméticos

- Os operadores aritméticos são usados para desenvolver operações matemáticas. A seguir apresentamos a lista dos operadores aritméticos do C:

Operador	Ação
+	Soma (inteira e ponto flutuante)
-	Subtração ou Troca de sinal (inteira e ponto flutuante)
*	Multiplicação (inteira e ponto flutuante)
/	Divisão (inteira e ponto flutuante)
%	Resto de divisão (de inteiros)
++	Incremento (inteiro e ponto flutuante)
--	Decremento (inteiro e ponto flutuante)

Operadores Aritméticos

- Exemplo,

```
# include <stdio.h>
void main( )
{
   int x, y, a, b;

   x = 5;     y = 2;
   a = x/y;
   b = x%y;
   printf("%d", x/y);      /*mostra 2*/
   printf("%d", x%y);    /*mostra 1*/
}
```

```
# include <stdio.h>
void main( )
{
   int x, y;

   x = 5;     y = 2;
   printf("%d", x/y);      /*mostra 2*/
   printf("%d", x%y);    /*mostra 1*/
}
```

printf("%d %d", x/y, x%y);

Operadores Aritméticos

x = x +1 ↔ ++x ou x++ e x = x -1 ↔ --x ou x--

Obs.: Há uma diferença quando os operadores x++ e ++x (x-- e --x) são usados em uma expressão. Quando um operador de incremento ou decremento precede seu operando, C executa a operação de incremento ou decremento antes de usar o valor do operando. Se o operador estiver após seu operando, C usará o valor do operando antes de incrementá-lo ou decrementá-lo.

Exemplo,

```
x = 10;
y = ++x;        /* y = 11, incrementar e atribuir */
x = 10;
y = x++;        /* y = 10, atribuir e incrementar */
```

Operadores Relacionais e Lógicos

- Os **Operadores Relacionais** do C realizam comparações entre variáveis. São eles:

Operador	Ação
>	Maior do que
>=	Maior ou igual a
<	Menor do que
<=	Menor ou igual a
==	Igual a
!=	Diferente de

Obs.: Os operadores relacionais retornam *verdadeiro* (1) ou *falso* (0).

Operadores Relacionais e Lógicos

- Para fazer operações com valores lógicos (verdadeiro e falso) temos os **Operadores Lógicos**:

Operador	Ação
&&	AND (E)
||	OR (OU)
!	NOT (NÃO)

- Usando os operadores relacionais e lógicos podemos realizar uma grande gama de testes. A tabela-verdade destes operadores é dada a seguir:

p	q	p&&q	p||q	!p
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

Operadores Relacionais e Lógicos

- Exemplo: No trecho de programa abaixo o **if** será executado, pois o resultado da expressão lógica é verdadeiro:

```
int i = 5, j =7;
if ( (i > 3) && ( j <= 7)  && ( i != j) )  j++;
```

V AND V AND V = V

Operadores Relacionais e Lógicos

- Exemplo: implementação do operador ou exclusivo (XOR)

```
#include <stdio.h>

int xor(int a, int b);

void main(void)
{
   printf("%d", xor(0, 0));
   printf("%d", xor(0, 1));
   printf("%d", xor(1, 0));
   printf("%d", xor(1, 1));
}

xor(int a, int b)
{
    return((a || b) && !(a && b));
}
```

Operadores Lógicos Bit a Bit

- O C permite que se faça operações lógicas "bit-a- bit" em números. Ou seja, neste caso, o número é representado por sua forma binária e as operações são feitas em cada bit dele.
- Imagine um número inteiro de 16 bits, a variável inteiro *i*, armazenando o valor 2. A representação binária de *i*, será: 0000000000000010 (quinze zeros e um único 1 na segunda posição da direita para a esquerda).

Operadores Lógicos Bit a Bit

- Poderemos fazer operações em cada um dos bits deste número. Por exemplo, se fizermos a negação do número 2 o número se transformará em 1111111111111101.
- As operações binárias ajudam programadores que queiram trabalhar com o computador em "baixo nível".
- As operações lógicas bit a bit só podem ser usadas nos tipos **char**, **int** e **long**.

Operadores Lógicos Bit a Bit

Operador	**Ação**
&	AND
|	OR
^	XOR (OR exclusivo)
~	NOT
>>	Deslocamento de bits a direita
<<	Deslocamento de bits a esquerda

- Os operadores &, |, ^ e ~ são as operações lógicas bit a bit.

p	q	p&q	p|q	p^q	~p
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

Operadores Lógicos Bit a Bit

- A forma geral dos operadores de deslocamento é:

  variável>>*número_de_deslocamentos*

  variável<<*número_de_deslocamentos*

- O *número_de_deslocamentos* indica o quanto cada bit irá ser deslocado. Por exemplo, para a variável *i* armazenando o número 2:

  i << 3;

  fará com que *i* agora tenha a representação binária: 0000000000010000, isto é, o valor armazenado em *i* passa a ser igual a 16.

Operadores Lógicos Bit a Bit

- Um deslocamento não é uma rotação. Ou seja, os bits deslocados são perdidos e zeros são colocados.
- Um deslocamento à direita efetivamente multiplica um numero 2 e um deslocamento à esquerda divide-o por 2

unsigned char x	**cada execução da sentença**	**valor de x**
x = 7;	00000111	7
x = x <<1;	00001110	14
x = x <<3;	01110000	112
x = x <<2;	11000000	192
x = x >>1;	01100000	96
x = x >>1;	00011000	24

Precedência dos Operadores

Maior precedência

( ) [ ] ->

! ~ ++ -- . -(unário) (cast) *(unário) &(unário) sizeof

* / %

+ -

<< >>

<<= >>=

== !=

&

^

|

&&

||

?

= += -= *= /=

Menor precedência

Expressões

- Expressões são combinações de variáveis, constantes e operadores.

Exemplos:

```
Anos = Dias/365.25;
i = i+3;
c= a*b + d/e;
c= a*(b+d)/e;
```

Expressões

- **Expressões que Podem ser Abreviadas**

  O C admite as seguintes equivalência, que podem ser usadas para simplificar expressões ou para facilitar o entendimento de um programa:

Expressão Original	Expressão Equivalente
x = x+k;	x += k;
x = x-k;	x -= k;
x = x*k;	x *= k;
x = x/k;	x /= k;
x = x>>k;	x >>= k;
x = x<<k;	x <<= k;
x = x&k;	x &= k

Expressões

- **Encadeando expressões: o operador virgula “,”**

  O operador , determina uma lista de expressões que devem ser executadas seqüencialmente. O valor retornado por uma expressão com o operador “,” é sempre dado pela expressão mais à direita.

  No exemplo abaixo:

```
x = (y=2, y+3);
```

o valor 2 vai ser atribuído a y, se somará 3 a y e o retorno (5) será atribuído à variável x . Pode-se encadear quantos operadores , forem necessários.

Expressões

- **Modeladores (Casts)**

Um modelador é aplicado a uma expressão. Ele força a mesma a ser de um tipo especificado. Sua forma geral é:

(tipo) expressão

```
#include <stdio.h>
 int main ()
 {
      int num;
      float f;
      num = 10;
      f = (float)num/7;
      printf ("%f", f);
      return(0);
}
```

Se não tivéssemos usado o modelador no exemplo acima o C faria uma divisão inteira entre 10 e 7. O resultado seria 1 (um) e este seria depois convertido para float mas continuaria a ser 1.0. Com o modelador temos o resultado correto.