Universidade de São Paulo
Brasil

Aula 13
Introdução à Linguagem C

MAC0216 - Técnicas de Programação I

Professores: Alfredo, Daniel, Fabio e Kelly

Departamento de Ciência da Computação
Instituto de Matemática e Estatística

1.
História

História

- Criada por Brian Kernighan e Dennis Ritchie em 1972 nos laboratórios da AT&T Bell
  - Construção de utilitários para Unix; re-escrita do kernel do Unix
- Padrão ANSI (*American National Standards Institute*)
  - ANSI C / C89 ou C90
- Padrão ISO (*International Standards Organization*)
  - C95, C99, C11, C18
- Compatibilidade com versões anteriores
  - Efeito colateral: muito código antigo por aí, que não se beneficia dos novos recursos da linguagem

C e C++ são linguagens diferentes!

- C++ foi criada a partir de uma versão bem inicial de C
- Estão se desenvolvendo de forma independente há mais de 30 anos
- Elas têm recursos em comum
  - Adotaram recursos uma da outra
  - Alguns recursos foram desenvolvidos conjuntamente para as duas

Linguagem C

- Linguagem estruturada
  - Problema complexo → decomposição em problemas mais simples
  - Programa é composto por módulos
  - Função é o módulo básico
- Tipagem estática
- Compilada

2.
Estrutura Geral

Esqueleto de um programa em C

```
/* bibliotecas usadas */
#include <stdio.h>

/* prototipos das funcoes */

int main()  {
        /* declaracao de variaveis */

        /* lista de comandos */

        return 0;    /* comandos terminam com ; */
}           /* blocos de comandos entre { }  */

 /* definicao das funcoes */
```

Sobre as funções

- Toda função tem um nome único (= identificador)
- A função **main** é necessária em todos os programas
  - Define o início da execução

Exemplo 1

A. Escreva uma função **`int fat (int n)`** que recebe como parâmetro um inteiro n e calcula o fatorial de n .
B. Utilizando a função do item anterior, faça uma função **`int comb (int m, int n)`** que recebe dois inteiros m e n , com m ≥ n , e calcula C(m,n) = m! / (n!(m-n)!)
C. Escreva um programa que leia dois inteiros m e n e exiba o valor da combinação C(m,n).

Exemplo 1

```c
#include <stdio.h>

/* Protótipo das funções */
int fat (int);
int comb (int, int);

int main () {
   int m, n;
     printf("Digite o valor de m: ");
   scanf("%d", &m);
   printf("\nDigite o valor de n: ");
   scanf("%d", &n);
   printf("\nC(%d,%d) = %d\n", m, n,
comb(m,n));
}
```

```c
/* Definição das funções */
int comb (int m, int n) {
   return fat(m) / (fat(n) * fat(m-n));
}

int fat (int n) {
   int fatorial = 1;
   int num = 1;

   while (num <= n) {
       fatorial *= num;
       num++;
   }

   return fatorial;
}
```

Declaração vs definição de funções

- Definir os protótipos das funções não é obrigatório, mas é uma boa prática de programação pois permite que você defina as funções em qualquer ordem

Exemplo 2

Dado um número inteiro n > 0, determinar o número harmônico Hn dado por

Hn = 1 + 1/2 + 1/3 + 1/4 + . . . + 1/n .

Imprima cada termo da sequência e o resultado final.

Dúvida: A ordem em que os termos são calculados e adicionados à soma interfere no resultado do programa?

Exemplo 2

```
    printf("Harmônico calculado do maior termo para o menor \n");
    for (int x = 1; x <= n; x++) {
        hn1 += 1.0/x;
        printf("Termo %d: 1/%d = %.10f, H1(%d) =  %.10f \n", x, x, 1.0/x, x, hn1);
    }

    printf("Harmônico calculado do menor termo para o maior \n");
    for (int x = n; x > 0; x--) {
        hn2 += 1.0/x;
        printf("Termo %d: 1/%d = %.10f, H2(%d) =  %.10f \n", x, x, 1.0/x, x, hn2);
    }
```

A soma calculada com menores termos para os maiores é a mais estável, por envolver somas entre números de valores mais próximos um do outro.

Para mais detalhes, ver: https://floating-point-gui.de/errors/propagation/

Tipos de Dados

Tipos de Dados

Class		Systematic name	Other name	Rank
Integers	Unsigned	`_Bool`	`bool`	0
		`unsigned char`		1
		`unsigned short`		2
		`unsigned int`	`unsigned`	3
		`unsigned long`		4
		`unsigned long long`		5
	[Un]signed	`char`		1
	Signed	`signed char`		1
		`signed short`	`short`	2
		`signed int`	`signed` or `int`	3
		`signed long`	`long`	4
		`signed long long`	`long long`	5
Floating point	Real	`float`		
		`double`		
		`long double`		
	Complex	`float _Complex`	`float complex`	
		`double _Complex`	`double complex`	
		`long double _Complex`	`long double complex`	

Fonte da tabela: Jens Gustedt, ***Modern C***, Manning, 2019.

Expressões

Expressões

- Operador de atribuição

  =
- Operadores matemáticos

  *, /, %, +, -
- Operadores lógicos

  && (and), || (or), ! (not)
- Operadores relacionais

  ==, !=, >, >=, <, <=

Atribuições

- Atribuição múltipla

```
int x, y;
x = y = 42;
```

- Atribuição na declaração de variável

```
int x = 0, y = 1;
```

- Abreviatura (para operadores matemáticos)

```
x /= 42;   // equivale a  x = x/42
```

- Abreviatura de incremento

  ++x, x++

- Abreviatura de decremento

  --x, x--

Entrada e Saída

Biblioteca <stdio.h>

- Impressão:

```
int printf (char const *format, ...);
```

- Leitura:

```
int scanf (const char *format, ...);
```

Especificação de formato para a *printf* pode ser composta por 5 partes:
%[FF][WW][.PP][LL]SS
(somente a SS é obrigatória)

FF	Flags	Special form of conversion
WW	Field width	minimum width
PP	Precision	
LL	Modifier	Select width of type
SS	Specifier	Select conversion

'd' or 'i'	Decimal	Signed integer
'u'	Decimal	Unsigned integer
'o'	Octal	Unsigned integer
'x' or 'X'	Hexadecimal	Unsigned integer
'e' or 'E'	[-]d.ddd e±dd, "scientific"	Floating point
'f' or 'F'	[-]d.ddd	Floating point
'g' or 'G'	generic e or f	Floating point
'a' or 'A'	[-]0xh.hhhh p±d, Hexadecimal	Floating point
'%'	'%' character	No argument is converted.
'c'	Character	Integer
's'	Characters	String
'p'	Address	**void*** pointer

Especificadores (SS) de formato para a função *printf*

Fonte da tabela: Jens Gustedt, ***Modern C***, Manning, 2019.

Character	Meaning	Conversion
"#"	Alternate form, such as prefix 0x	"aAeEfFgGoxX"
"0"	Zero padding	Numeric
"-"	Left adjustment	Any
"␣"	'␣' for positive values, '-' for negative	Signed
"+"	'+' for positive values, '-' for negative	Signed

Flags (FF) de formato para a função *printf*

Character	Type
"hh"	**char** types
"h"	**short** types
""	**signed**, **unsigned**, **float**, **char** arrays and strings
"l"	**long** integer types, **double**, **wchar_t** characters and strings
"ll"	**long long** integer types
"j"	**intmax_t**, **uintmax_t**
"z"	**size_t**
"t"	**ptrdiff_t**
"L"	**long double**

Modificadores (LL) de formato para a função *printf*

Fonte da tabela: Jens Gustedt, ***Modern C***, Manning, 2019.

Especificação de formato para a *scanf* pode ser composta por 4 partes:
%[XX][WW][LL]SS
(somente a SS é obrigatória)

Especificadores (SS) de formato para a função *scanf*

XX	*	Assignment suppression
WW	Field width	Maximum number of input characters
LL	Modifier	Select width of target type
SS	Specifier	Select conversion

SS	Conversion	Pointer to	Skip space
'd'	Decimal	Signed type	Yes
'i'	Decimal, octal, or hex	Signed type	Yes
'u'	Decimal	Unsigned type	Yes
'o'	Octal	Unsigned type	Yes
'x'	Hexadecimal	Unsigned type	Yes
'aefg'	Floating point	Floating point	Yes
'%'	'%' character	No assignment	No
'c'	Characters	**char**	No
's'	Non-whitespace	**char**	Yes
'['	Scan set	String	No
'p'	Address	**void**	Yes
'n'	Character count	Signed type	No

Fonte da tabela: Jens Gustedt, ***Modern C***, Manning, 2019.

Character	Type
"hh"	**char** types
"h"	**short** types
""	**signed**, **unsigned**, **float**, **char** arrays and strings
"l"	**long** integer types, **double**, **wchar_t** characters and strings
"ll"	**long long** integer types
"j"	**intmax_t**, **uintmax_t**
"z"	**size_t**
"t"	**ptrdiff_t**
"L"	**long double**

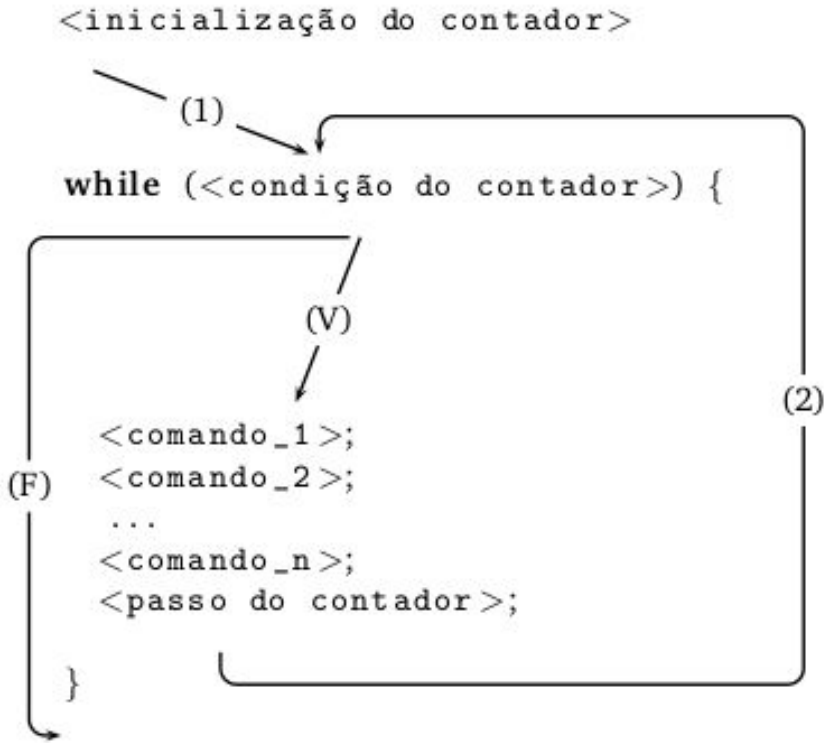
Modificadores (LL) de formato para a função *scanf*

Fonte da tabela: Jens Gustedt, ***Modern C***, Manning, 2019.

Controle de Fluxo - Laços

Comandos *while*, *do while* e *for*

Laço while

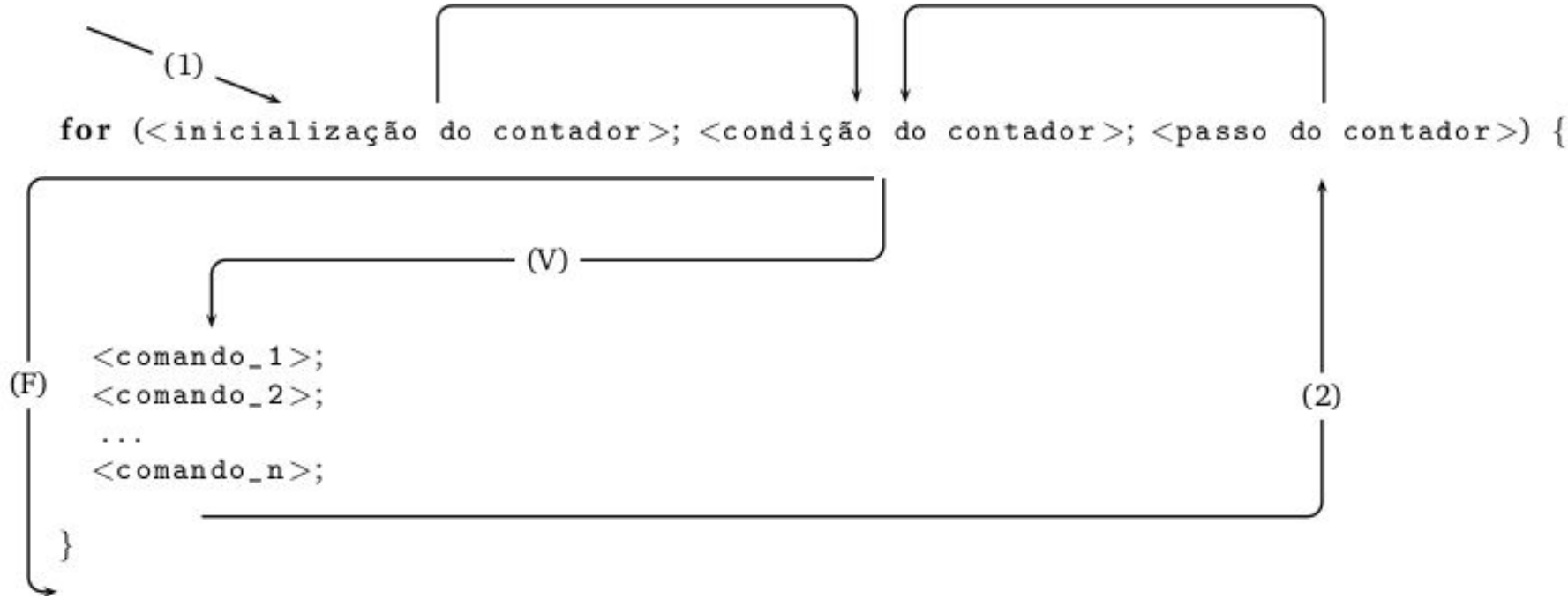

<inicialização do contador>
(1)
while (<condição do contador>) {
(V)
<comando_1>;
<comando_2>;
...
<comando_n>;
<passo do contador>;
}
(2)
(F)

Laço do while

```
do {
    <comando_1>;
    <comando_2>;
    ...
    <comando_n>;
} while (<condição>);
```

- O laço é sempre executado pelo menos uma vez

Laço for


(1)
for (<inicialização do contador>; <condição do contador>; <passo do contador>) {
(V)
<comando_1>;
<comando_2>;
...
<comando_n>;
(2)
(F)
}

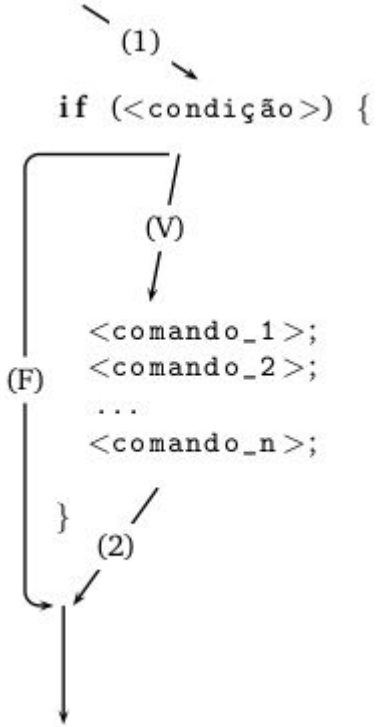
Delimitação de blocos

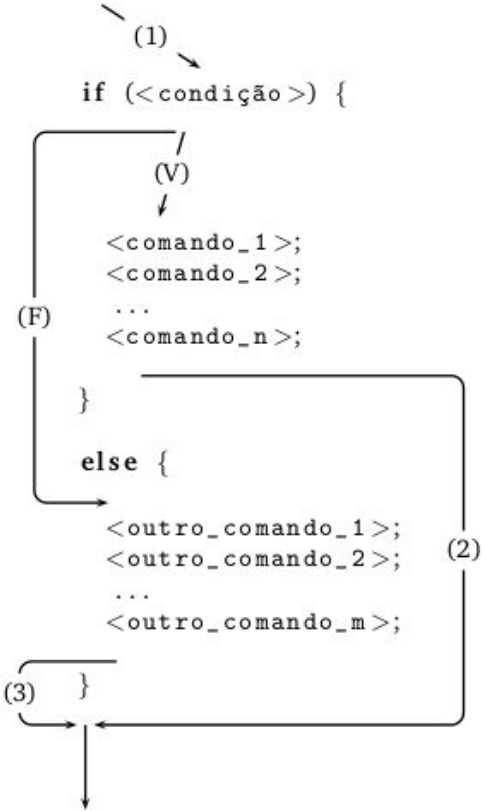
- Quando um bloco de comandos tem apenas um comando, não é necessário delimitá-lo com { }

Controle de Fluxo - Seleção

Comandos *if-else* e *switch*

Seleção com if-else





Seleção com switch

```
switch (<expressão>)
{
   case <constante_1>:
     <comandos_1>;
   case <constante_2>:
     <comandos_2>;
   ...
   default
     <comandos_n>;
}
```

- A expressão e as constantes precisam ser um valor integer ou um caractere
- Uma vez que a execução entrou no bloco, ela vai continuar até encontrar um comando *break* ou o fim do bloco

```c
#include <stdio.h>
int main() {
 int number = 2;
 switch (number)  {
 case 1:
 case 2:
 case 3:
   printf("Um, doi ou três.\n");
   break;
 case 4:
 case 5:
   printf("Quatro e cinco.\n");
   break;
 default:
   printf("Maior que cinco.\n");
 }
}
```

Passagem de Parâmetro

Por valor vs Por endereço (ponteiro)

Ponteiros

- Declaração
  - Ex.: int *variavel_ponteiro;
- Operador &<variavel>: devolve o endereço da variável
- Operador *<variável_ponteiro>: acessa a posição de memória apontada pela variável ponteiro

Exemplo 3

Faça um função que recebe como parâmetros de entrada três reais a, ($a \neq 0$), b e c e resolve a equação de 2o. grau $ax^2+bx+c=0$, devolvendo as raízes em dois ponteiros $*x1$ e $*x2$.

Esta função ainda deve devolver via

return o valor -1 se a equação não tem raízes reais, 0 se tem somente uma raiz real e 1 se a equação tem duas raízes reais distintas.

Exemplo 3

```c
#include <math.h>

int segundo_grau(float a, float b, float c, float *x1, float *x2) {
    float delta = b*b - 4*a*c;

    if (delta < 0)
        return -1;

    *x1 = (-b + sqrt(delta)) / (2*a);
    *x2 = (-b - sqrt(delta)) / (2*a);

    if (delta > 0)
        return 1;
    return 0;
}
```

Exemplo 3 - continuação

```
int main() {
   float a, b, c, r1, r2, tem_raiz;
   printf("Entre com os coeficientes a, b e c: ");
   scanf("%f %f %f", &a, &b, &c);

   tem_raiz = segundo_grau(a, b, c, &r1, &r2);
   if (tem_raiz < 0)
       printf("Equação não tem raízes reais.\n");
   else if (tem_raiz == 0)
       printf("Equação tem somente uma raiz: %f.\n", r1);
   else
       printf("Equação tem duas raízes: %f e %f.\n", r1, r2);
   return 0;
}
```

Vetores

Vetores

- Estruturas indexadas
  - Índice é um número natural
  - Primeira posição tem índice 0
- Armazenam elementos de um mesmo tipo
- Declaração:

<tipo_do_vetor> <nome_do_vetor> [<tamanho_do_vetor>];

  - <tamanho_do_vetor> deve ser um valor constante
- Tamanho declarado vs tamanho usado

Vetores - Erros comuns

- Acesso a posições inválidas
  - Índices negativos
  - Índices maiores que o tamanho do vetor

Exemplo 4

A) Escreva uma função com protótipo

int acha (float V[MAX], int n, float x);

que devolve a posição em que o real x ocorre no vetor V ou devolve -1 se x não aparece no vetor. O número de elementos do vetor é n.

B) Escreva uma função com protótipo

int insere (float V[MAX], int n, float x);

que insere x na última posição do vetor V e retorna o novo valor de n.

C) Dada uma sequência de n números reais, imprimi-la eliminando as repetições.

Exemplo 4

```c
#include <stdio.h>
#define MAX 100

int acha(float V[MAX], int n, float x){
   int i = 0;

   while (i < n && V[i] != x)
       i++;
   if (i != n)
       return i;
   return -1;
}

int insere (float V[MAX], int n, float x) {
   V[n]= x;
   return n+1;
}
```

```c
int main(){
   float seq[MAX], num;
   int n, i, k = 0;
   printf("Digite o número de elementos da sequência: ");
   scanf("%d", &n);
   for (i = 0; i < n; i++){
       printf("Digite o %do. número da sequência: ", i+1);
       scanf("%f", &num);
       if (acha(seq,k,num) == -1)
           k = insere(seq,k,num);
   }
   printf("\nA sequência sem repetições é: \n");
   for (i = 0; i < k; i++)
       printf("%f   ", seq[i]);
   printf("\n");

   return 0;
}
```

Macros como constantes

```
#define <NOME_DA_CONSTANTE> <valor>
Ou
#define <NOME_DA_CONSTANTE> <expressão>
```

- **#define** é uma diretriz para definição de macros
  - Mecanismo que faz substituições textuais no código do programa
  - Substituição é feita pelo pré-processador do código, antes da compilação

Referências Bibliográficas

- Carlos Hitoshi Morimoto, Ronaldo Fumio Hashimoto, ***Introdução a Ciência da Computação em C*** (Apostila)

  https://www.ime.usp.br/~hitoshi/introducao/index.html
- Jens Gustedt, ***Modern C***, Manning, 2019.

  https://modernc.gforge.inria.fr/
- E. Roberts, ***The Art and Science of C***, Addison-Wesley, 1995.
- Paulo Feoffilof, ***Projeto de Algoritmos em C***, ver links na seção "Recursos da linguagem C:" https://www.ime.usp.br/~pf/algoritmos/
- Exercícios

  https://www.ime.usp.br/~macmulti/