TE091
Programação Orientada a Objetos
Engenharia Elétrica

Revisão Rápida de Programação em C

Prof. Carlos Marcelo Pedroso

2015

Revisão – Linguagem C

- Características principais
  - Modularidade: uso de procedimentos e funções, importante característica de liguagens estruturadas;
  - Portabilidade: disponibilidade em múltiplas plataformas;
  - Velocidade: a linguagem C é uma linguagem de nível intermediário, o que favorece a eficiência do código (em velocidade e tamanho do executável)

- Aula de hoje
  - Revisão rápida de programação em C
    - Fazer exercícios de revisão para próxima semana.

Acknowledgement: os slides foram baseados no material de Donghui Zhang, Lewis Girod e Dan Negrut

Sintaxe e Hello World

O que isso significa? Anote de acordo com a aula

Comentário. Em C++ ou java, pode ser usado //

A função main() é a primeira função chamada quando o programa é executado. Anote como argc e argv são utilizados

Blocos de código são marcados por { … }

```
#include <stdio.h>
/* The simplest C Program */
int main(int argc, char **argv)
{
  printf("Hello World\n");
  return 0;
}
```

Função retorna 0

Escopo de variáveis

Cada variável é definida em um escopo. A não pode ser referenciada pelo seu nome for a de seu escopo scope.

O bloco é delimitado pelos símbolos { }.

- O escopo dos parâmetros de uma função compreende o bloco da função
- O escopo de variáveis definidas dentro de blocos tem validade no próprio bloco
- O escopo de variáveis definidas for a dos blocos de função tem validade global, para todo o arquivo do fonte

```
void p(char x)
{
                /* p,x */
  char y;
                /* p,x,y */
  char z;
                /* p,x,y,z */
}
                /* p */
char z;
                /* p,z */

void q(char a)
{
  char b;
                /* p,z,q,a,b
*/
                                char b?
  {
    char c;
                /* p,z,q,a,b,c
*/                              legal?
  }

  char d;
  /* p,z,q,a,b,d (not c)
*/
}

/* p,z,q */
```

Operadores Matemáticos e de Comparação

```
== equal to
<  less than
<= less than or equal
>  greater than
>= greater than or equal
!= not equal
&& logical and
|| logical or
!  logical not
```

```
+  plus
-   minus
*  mult
/  divide
%  modulo
```

```
&  bitwise and
|  bitwise or
^  bitwise xor
~  bitwise not
<< shift left
>> shift right
```

Anote a explicação e os exemplos mostrados durante a aula!

Operadores matemáticos

```
x = y    assign y to x
x++      post-increment x
++x      pre-increment x
x--      post-decrement x
--x      pre-decrement x
```

```
x += y   assign (x+y) to x
x -= y   assign (x-y) to x
x *= y   assign (x*y) to x
x /= y   assign (x/y) to x
x %= y   assign (x%y) to x
```

Diferença entre ++x and x++

```
int x=5;
int y;
y = ++x;
/* x == 6, y == 6 */
```

```
int x=5;
int y;
y = x++;
/* x == 6, y == 5 */
```

Não confundir “=“ com “==“!

```
int x=5;
if (x==6)   /* false */
{
  /* ... */
}
/* x is still 5 */
```

```
int x=5;
if (x=6)   /* always true
*/
{
  /* x is now 6 */
}
/* ... */
```

Como funciona o processo de compilação?

```
#include <stdio.h>
/* The simplest C Program
*/
int main(int argc, char
**argv)
{
  printf("Hello World\n");
  return 0;
}
```

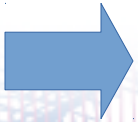
Pré processamento

```
__extension__ typedef  unsigned long long int
  __dev_t;
__extension__ typedef  unsigned int
__uid_t;
__extension__ typedef  unsigned int
__gid_t;
__extension__ typedef  unsigned long int
__ino_t;
__extension__ typedef  unsigned long long int
  __ino64_t;
__extension__ typedef  unsigned int
__nlink_t;
__extension__ typedef  long int   __off_t;
__extension__ typedef  long long int
__off64_t;
extern void flockfile (FILE *__stream)  ;
extern int ftrylockfile (FILE *__stream)  ;
extern void funlockfile (FILE *__stream)  ;
int main(int argc, char **argv)
{
  printf("Hello World\n");
  return 0;
}
```

No pré processamento o código fonte é expandido incluindo definições padrão. Todas as linhas iniciando com # são tratadas pelo pré processador

- Include files are "pasted in" (#include)
- Macros are "expanded" (#define)
- Comments are stripped out (/* */ , //)
- Continued lines are joined (\ )

Compilador

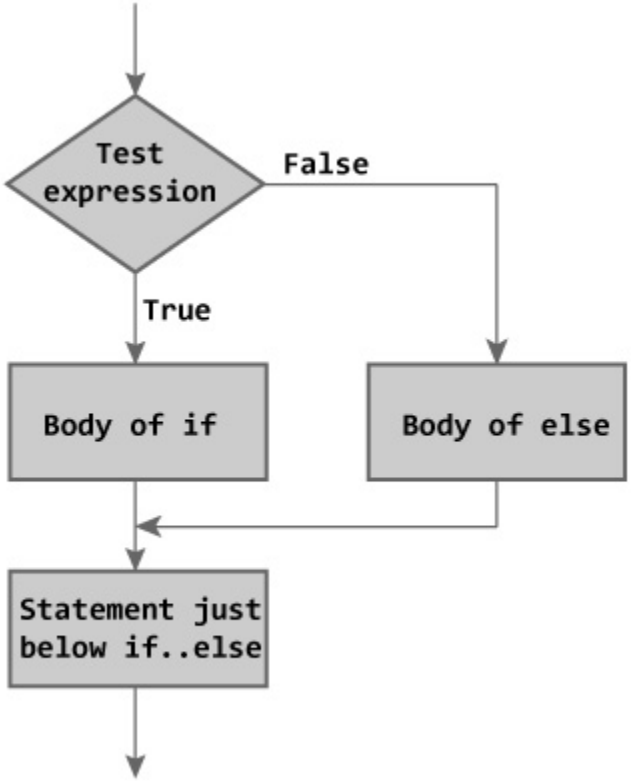


Programa

O compilador converte o programa fonte para código binário na máquina alvo. Bibliotecas com código já compilado podem ser incluídos.
Anote na aula: qual a diferença entre #include<> e uma bilioteca

Blocos de Desvio Condicional: if

```
if (test expression) {
     Bloco de comandos a ser executado
     caso o teste seja verdadeiro;
}
else {
    Bloco de comandos a ser executado
    caso o teste seja falso;
}
```



Blocos de Desvio Condicional: switch … case

```
switch (n) {
case constante1: bloco código;
                 break;
case constante2: bloco código;
                 break;
. . .
default: bloco código caso n
         não seja igual a nenhuma
         das constantes dos cases
         anteriores
}
```

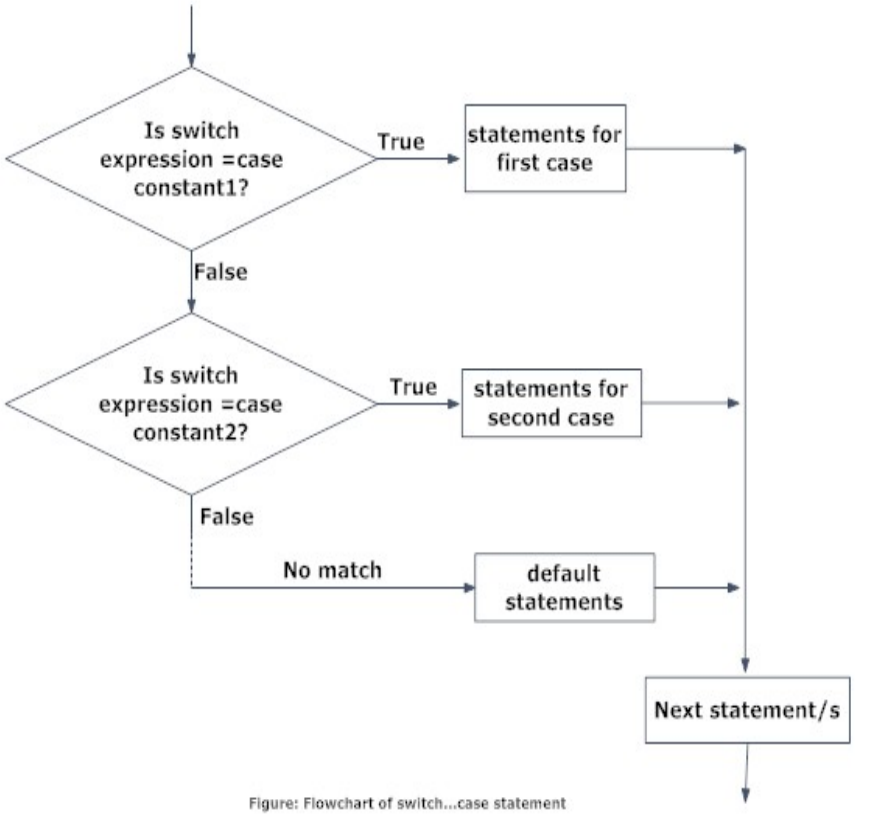


Figure: Flowchart of switch...case statement

Blocos de Desvio Condicional: while

```
while (teste) {
    Bloco para ser executado
    Caso teste seja verdadeiro;
}
```

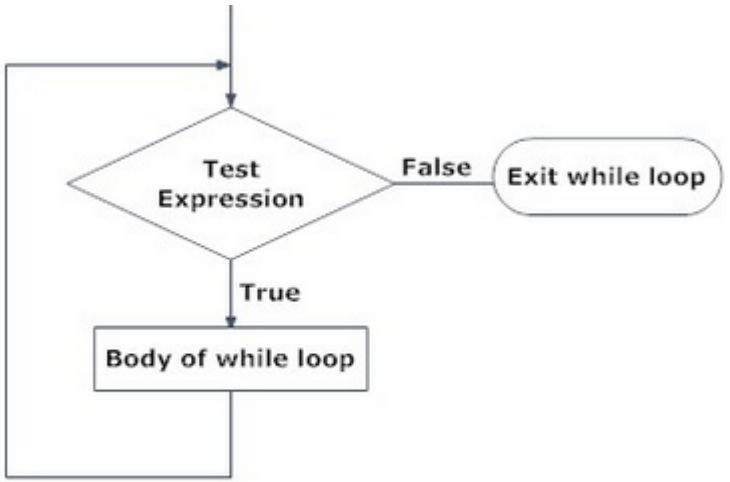


Figure: Flowchart of while loop

Blocos de Desvio Condicional: do … while

```
do {
    Bloco de código;
}
while (teste);
```

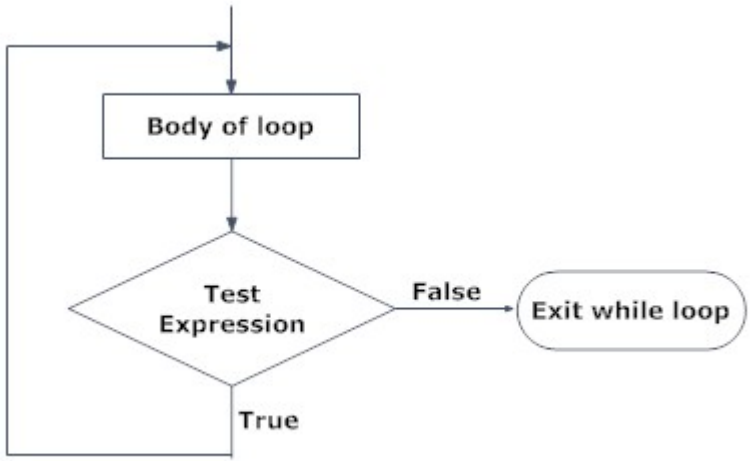
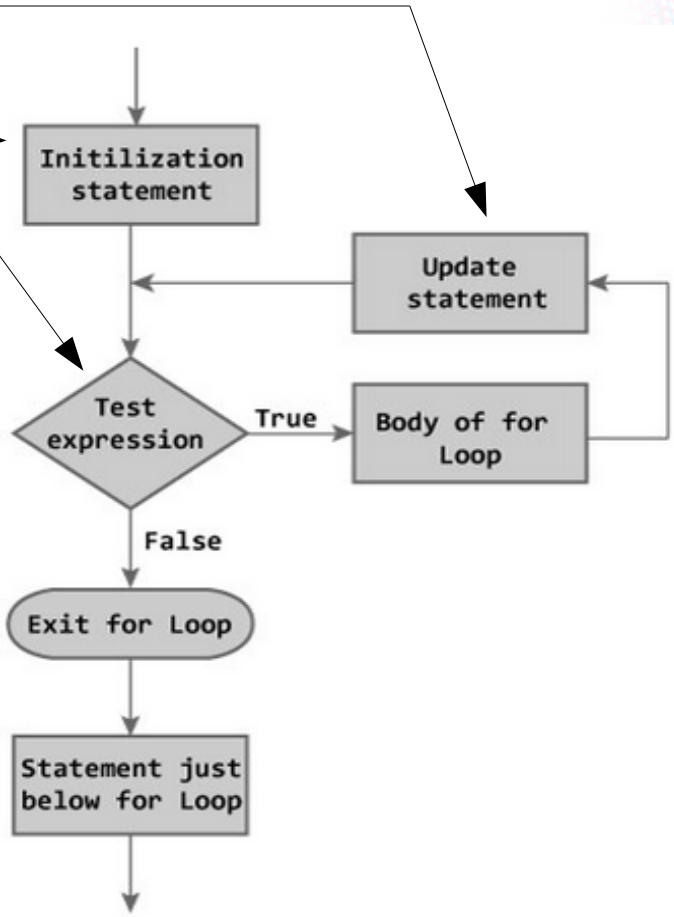


Figure: Flowchart of do...while loop

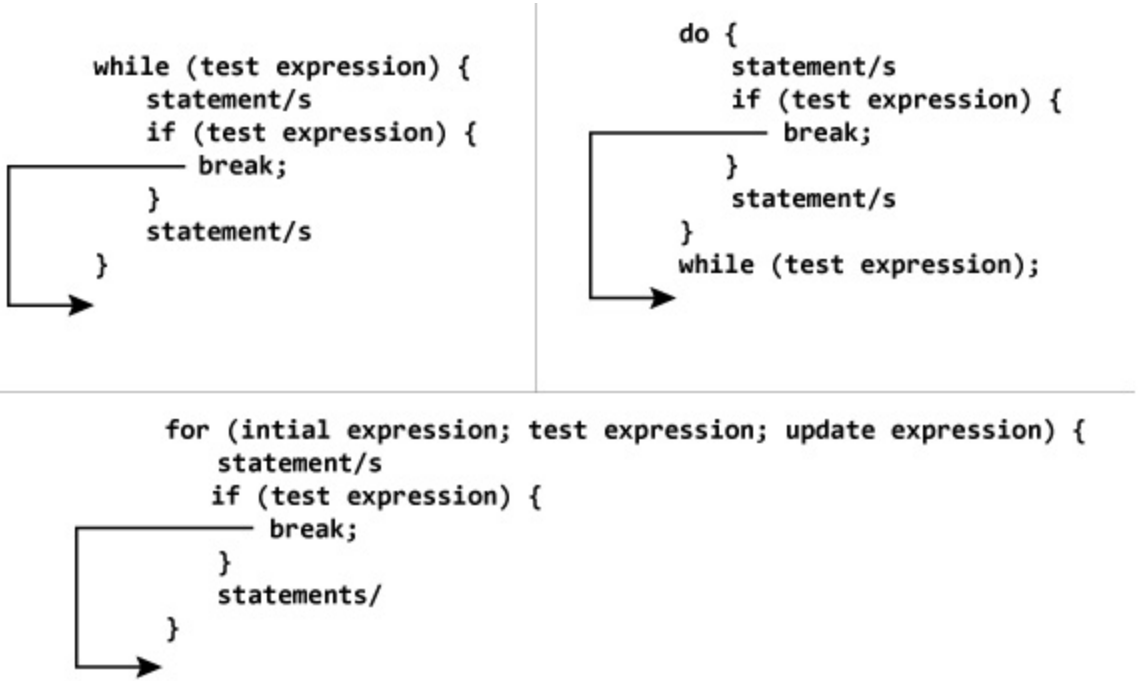
Blocos de Desvio Condicional: for

```
for(inicialização; teste; atualização)
{
     Bloco de código a ser executado
}
```



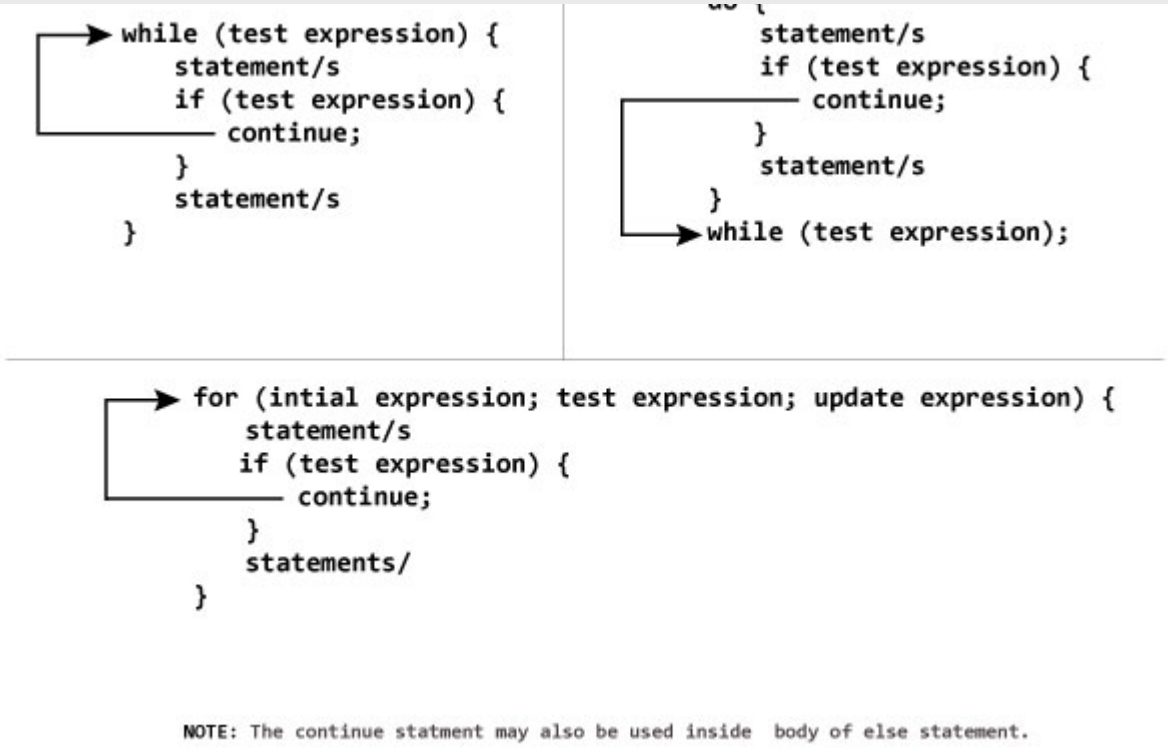
Uso do break

A diretiva break interrompe imediatamente a execução do loop mais interno.



break e continue

A diretiva continue interrompe a execução dos comandos do bloco e executa o teste do loop mais interno.



Memória

Memória: semelhate a uma grande tabela onde os bytes são armazenados

Cada posição tem seu Endereço.
Um valor pode ser armazenado em cada posição.

Ver Exemplo

O tipo indica o número de posições que uma variável ocupa e como a informação é usada

`char`	Um byte
`char[10]`	Vetor de 10 bytes
`int`	4 bytes
`float`	4 bytes em ponto flutuante

End	Valor
...	...
101	
102	84
103	69
104	83
105	84
106	69
107	0
108	
...	

ASCII
'T'
'E'
'S'
'T'
'E'
0

Ponteiros

- Anote os conceitos de acordo com a explicação na aula
  - Endereços de variáveis
  - Endereços de vetores
  - Ponteiros
  - '&' e '*'

Funções

- Parâmetros: uso da pilha;
- Variáveis locais: uso da pilha;
- Retorno de funções em C;
- Passagem de parâmetros por referência;
- Recursividade em C;
- **Anote os exemplos e explicações na aula.**

Alocação Dinâmica de Memória (Heap)

- Permite que um programa determine a quantidade de memória necessária em tempo de execução e realize a alocação em tempo de execução
  - No entanto, o programador é responsável por desalocar a memória ao final do uso.
  - Um grande número de problemas ocorre quando o programador comete erros
  - Ver exemplo de alocação dinâmica (durante a aula).
- A região de memória onde é realizada a alocação dinâmica de memória é chamada de Heap