

MC-102 — Aula 06

Teste de Mesa

Comandos Repetitivos

Instituto de Computação – Unicamp

Primeiro Semestre de 2012

Roteiro

- 1 Simulação de código
- 2 Comandos Repetitivos
- 3 `while (condicao) { comandos }`
- 4 `do { comandos } while (condicao);`
- 5 Laços e comandos `continue` e `break`

Introdução

- Às vezes, acontece de programarmos um código, porém ele não faz o que esperávamos que fizesse.
- Isso acontece por vários motivos, entre os quais destacam-se:
 - ▶ Erros de programação: instruções escritas erradas.
 - ▶ Erros da nossa lógica: o conjunto de passos pensados que parecia resolver o problema na realidade não cobre todas as situações.
- Eventualmente, simplesmente olhar o código pode não trazer a tona o erro.
- Por isso, utiliza-se uma técnica de simulação do código
 - ▶ Pode ser automatizada (utilizando um *debugger*)
 - ▶ Pode ser feita manualmente, utilizando papel e caneta.

Simulação Manual

- Bem simples: Existem apenas 2 passos.
 - ▶ “Alocação” dos espaços de variáveis
 - ▶ “Execução” de uma instrução de cada vez.
- Alocação de memória:
 - ▶ Ex. Suponha o código:
 1. `int divisor,dividendo;`
 2. `float resultado;`
- Após “executar” a linha 1

Tipo	<code>int</code>	<code>int</code>
Nome	<code>divisor</code>	<code>dividendo</code>
Valor	?	?

Simulação Manual

- Bem simples: Existem apenas 2 passos.
 - ▶ “Alocação” dos espaços de variáveis
 - ▶ “Execução” de uma instrução de cada vez.
- Alocação de memória:
 - ▶ Ex. Suponha o código:
 1. `int divisor,dividendo;`
 2. `float resultado;`
- Após “executar” a linha 2

Tipo	int	int	float
Nome	divisor	dividendo	resultado
Valor	?	?	?

Simulação Manual

- Execução em memória:
 - ▶ Ex. Suponha o código:
 1. `int divisor,dividendo;`
 2. `float resultado;` ← Último executado
 3. `divisor=10;` ← Próximo Comando
 4. `dividendo=13;`
 5. `resultado = dividendo / divisor;`
- Após “executar” a linha 2

Tipo	int	int	float
Nome	divisor	dividendo	resultado
Valor	?	?	?

Simulação Manual

- Execução em memória:
 - ▶ Ex. Suponha o código:
 1. `int divisor,dividendo;`
 2. `float resultado;`
 3. `divisor=10;` ← Último executado
 4. `dividendo=13;` ← Próximo Comando
 5. `resultado = dividendo / divisor;`
- Após “executar” a linha 3

Tipo	int	int	float
Nome	divisor	dividendo	resultado
Valor	? 10	?	?

Simulação Manual

- Execução em memória:
 - ▶ Ex. Suponha o código:
 1. `int divisor,dividendo;`
 2. `float resultado;`
 3. `divisor=10;`
 4. `dividendo=13;` ← Último executado
 5. `resultado = dividendo / divisor;` ← Próximo Comando
- Após “executar” a linha 4

Tipo	int	int	float
Nome	divisor	dividendo	resultado
Valor	? 10	? 13	?

Simulação Manual

- Execução em memória:
 - ▶ Ex. Suponha o código:
 1. `int divisor,dividendo;`
 2. `float resultado;`
 3. `divisor=10;`
 4. `dividendo=13;`
 5. `resultado = dividendo / divisor;` ← Último executado
- Após “executar” a linha 5

Tipo	int	int	float
Nome	divisor	dividendo	resultado
Valor	?	?	?
	10	13	1.0

Simulação Manual

- Execução em memória:
 - ▶ Ex. Suponha o código:
 1. `int divisor,dividendo;`
 2. `float resultado;`
 3. `divisor=10;`
 4. `dividendo=13;`
 5. `resultado = dividendo / divisor;` ← Último executado
- Término da execução (não há mais comandos)

Tipo	int	int	float
Nome	divisor	dividendo	resultado
Valor	?	?	?
	10	13	1.0

Simulação Manual

- Execução em memória:

- ▶ Ex. Suponha o código (corrigido):

- 1. `int divisor,dividendo;`

- 2. `float resultado;`

- 3. `divisor=10;`

- 4. `dividendo=13;`

- 5. `resultado = (float)dividendo / (float)divisor;`

- Execução completa

Tipo	int	int	float
Nome	divisor	dividendo	resultado
Valor	?	?	?
	10	13	1.3

Comandos Repetitivos

- Até agora, vimos como escrever programas capazes de executar comandos de forma linear, e, se necessário, tomar decisões com relação a executar ou não um bloco de comandos.
- Entretanto, eventualmente é necessário executar um bloco de comandos várias vezes para obter o resultado.

Exemplo

Calcule a divisão inteira de dois números usando apenas soma e subtração

Solução

- Duas variáveis auxiliares: temporario, contador
 - 1 temporario=dividendo;
 - 2 contador=0;
 - 3 Enquanto *temporario* > *divisor*
 - 1 temporario = temporario - divisor
 - 2 contador = contador + 1
 - 4 Exiba contador

Por que?

Contador equivale a divisão inteira de dividendo por divisor

Introdução

- Será que dá pra fazer com o que já temos?
- Ex.: Programa que imprime todos os números de 1 a 4

```
printf("1");  
printf("2");  
printf("3");  
printf("4");
```

Introdução

- Ex.: Programa que imprime todos os números de 1 a 100

```
printf("1");  
printf("2");  
printf("3");  
printf("4");  
/*repete 95 vezes a linha acima*/  
printf("100");
```

Introdução

- Ex.: Programa que imprime todos os números de 1 a n (dado)

```
printf("1");
if (n>=2)
    printf("2");
if (n>=3)
    printf("3");
/*repete 96 vezes o bloco acima*/
if (n>=100)
    printf("100");
```



```
while (condicao) { comandos }
```

- Estrutura:

```
while ( condicao )  
    comando;
```

- Ou:

```
while ( condicao ){  
    comandos;  
}
```

- Enquanto a condição for verdadeira ($\neq 0$), ele executa o(s) comando(s);
- Passo 1: Testa condição. Se condição for verdadeira vai para Passo 2.
- Passo 2.1: Executa comandos;
- Passo 2.2: Volta para o Passo 1.

Imprimindo os 100 primeiros números inteiros

```
int i=1;
while (i<=100)
{
    printf("%d ",i);
    i++;
}
```

Imprimindo os n primeiros números inteiros

```
int i=1,n;
scanf("%d",&n);
while (i<=n)
{
    printf("%d ",i);
    i++;
}
```

Imprimindo as n primeiras potências de 2

```
int i=1,n,pot=2;
scanf("%d",&n);
while (i<=n)
{
    printf("2^%d = %d ",i,pot);
    i++;
    pot = pot*2;
}
```

```
while (condicao) { comandos }
```

- 1. O que acontece se a condição for falsa na primeira vez?
`while (a!=a) a=a+1;`
- 2. O que acontece se a condição for sempre verdadeira?
`while (a==a) a=a+1;`

```
while (condicao) { comandos }
```

- 1. O que acontece se a condição for falsa na primeira vez?

```
while (a!=a) a=a+1;
```

R: Ele nunca entra na repetição (*loop*).

- 2. O que acontece se a condição for sempre verdadeira?

```
while (a==a) a=a+1;
```

R: Ele entra na repetição e nunca sai (*loop* infiniiiiiiiiiiiiiiiiiiiiii
ii
ii.....to)

```
do { comandos } while (condicao);
```

- Estrutura:

```
do
    comando;
while ( condicao );
```

- Ou:

```
do{
    comandos;
}while ( condicao );
```

- Diferença do `while`: Sempre executa comandos na primeira vez. Teste condicional é feito por último.
- Passo 1: Executa comandos;
- Passo 2: Testa condição. Se for verdadeira vai para Passo 1.

Imprimindo os 100 primeiros números inteiros

```
int i;  
i=1;  
do{  
    printf("\n %d",i);  
    i = i+1;  
}while(i<= 100);
```


Imprimindo os n primeiros números inteiros

```
int i, n;  
i=1;  
scanf("%d",&n);  
do{  
    printf("\n %d",i);  
    i++;  
}while(i<=n);
```

- O que acontece quando digita 0 ($n=0$)?

Imprimindo as n primeiras potências de 2

```
int i, n, pot;
pot = 2;
i = 1;
scanf("%d",&n);
do{
    printf("\n %d",pot);
    pot = pot *2;
    i++;
}while(i<= n);
```

- O que acontece quando digita 0 ($n=0$)?

```
for (inicio ; condicao ; passo) { comandos ; }
```

- Estrutura:

```
for (inicio ; condicao ; passo)
    comando ;
```

- Ou:

```
for (inicio ; condicao ; passo) {
    comandos;
}
```

- Início: valor inicial da variável de controle do laço
- Condição: Executa enquanto a condição for verdadeira
- Passo: valor da variável de controle do laço no próximo passo

```
for (inicio ; condicao ; passo) { comandos ; }
```

- Passo 1: Executa comandos em “inicio”.
- Passo 2: Testa condição. Se for verdadeira vai para passo 3.
- Passo 3.1: Executa comandos
- Passo 3.2: Executa comandos em “passo”.
- Passo 3.2: Volta ao Passo 2.

o **for** é equivalente a seguinte construção utilizando o **while**:

```
inicio;  
while(condicao){  
    comandos;  
    passo;  
}
```

Imprimindo os 100 primeiros números inteiros

```
int i;  
for(i=1; i<= 100; i=i+1){  
    printf("\n %d",i);  
}
```

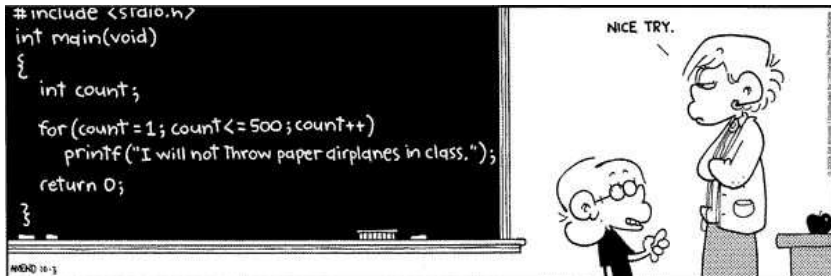
Imprimindo os n primeiros números inteiros

```
int i, n;  
scanf("%d",&n);  
for(i=1; i<=n; i++){  
    printf("\n %d",i);  
}
```

Imprimindo as n primeiras potências de 2

```
int i, n, pot;
pot = 2;
scanf("%d",&n);
for(i=1; i<= n; i++){
    printf("\n %d",pot);
    pot = pot *2;
}
```

I'll not throw paper airplanes in class



Laços e o comando break

O comando **break** faz com que a execução de um laço seja terminada, passando a execução para o o próximo comando depois do final do laço.

```
int i;
for(i = 1; i<= 10 ; i++){
    if(i >= 5)
        break;
    printf("%d\n",i);
}
printf("Terminou o laço");
```

O que será impresso?

Laços e o comando break

Assim como a “condição” em laços, o comando **break** é utilizado em situações de parada de um laço.

```
int i;
for(i = 1; ; i++){
    if(i > 10)
        break;
    printf("%d\n",i);
}
```

é equivalente a:

```
int i;
for(i = 1; i<=10 ; i++){
    printf("%d\n",i);
}
```

Laços e o comando continue

O comando **continue** faz com que a execução de um laço seja alterada para final do laço.

```
int i;
for(i = 1; i<= 10 ; i++){
    if(i == 5)
        continue;
    printf("%d\n",i);
}
printf("Terminou o laço");
```

O que será impresso?

Laços e o comando continue

O comando **continue** é utilizado em situações onde comandos dentro do laço só serão executados caso alguma condição seja satisfeita. Imprimindo área de um círculo, mas apenas se raio for par (e entre 1 e 10).

```
int r;
double area;
for(r = 1; r<= 10 ; r++){
    if( (r % 2) != 0)
        continue;
    area = 3.1415*r*r;
    printf("%lf\n",area);
}
```

Mas note que poderíamos escrever algo mais simples:

```
int r;
double area;
for(r = 2; r<= 10 ; r = r+2){
    area = 3.1415*r*r;
    printf("%lf\n",area);
}
```

Exercício

- Faça um programa que lê um número n e imprima os valores entre 2 e n , que são divisores de n .

Exercício

- Faça um programa que imprima um menu de 4 pratos na tela e uma quinta opção para sair do programa. O programa deve imprimir o prato solicitado. O programa deve terminar quando for escolhido a quinta opção.

Exercício

- Faça um programa que lê um número n e que compute e imprima o valor

$$\sum_{i=1}^n i.$$

OBS: Não use fórmulas como a da soma de uma P.A.

Exercício

- Faça um programa que lê um número n e imprima os valores

$$\sum_{i=1}^j i$$

para j de 1 até n , um valor por linha.