

Estruturas de Repetição - Parte 2

Algoritmos e Programação de Computadores – ABI/LFI/TAI



**INSTITUTO
FEDERAL**
Brasília

Prof. Daniel Saad Nogueira
Nunes

IFB – Instituto Federal de Brasília,
Campus Taguatinga



Sumário

- 1 Introdução
- 2 Laços Infinitos
- 3 Interrupção
- 4 Exemplos
- 5 Laços Aninhados



Sumário

1 Introdução



Introdução

- Vimos na aula anterior as estruturas de repetição suportadas pela linguagem C.
- Nesta aula veremos uma série de exemplos em como aplicar as estruturas vistas para resolver diversos problemas.
- Usaremos diferentes padrões de resolução de problemas, os quais são úteis conhecer na hora de resolver problemas novos, pois os mesmos padrões poderão se aplicar.
- Também veremos os comandos de interrupção de fluxo.
- Além disso, abordaremos a questão de como escrever laços infinitos.



Sumário

2 Laços Infinitos



Laços Infinitos

- Nem todo o programa precisa parar de executar.
- Temos diversos exemplos de sistemas que não devem parar de executar, a menos que um comando explícito seja dado.
- Exemplo: Sistema Operacional.
- Podemos escrever laços que nunca param de executar, para que efetue-se um processamento infinito.
- Basta que a condição de parada nunca seja falsa!



Laços Infinitos: While

```
1  #include <stdio.h>
2
3  int main(void) {
4      int i = 1;
5      while (1) {
6          printf("%d\n", i);
7          i++;
8      }
9      return 0;
10 }
```



Laços Infinitos: For

```
1  #include <stdio.h>
2
3  int main(void) {
4      int i;
5      for (i = 1;; i++) {
6          printf("%d\n", i);
7      }
8      return 0;
9  }
```



Laços Infinitos: For

```
1  #include <stdio.h>
2
3  int main(void) {
4      int i = 1;
5      do {
6          printf("%d\n", i);
7          i++;
8      } while (1);
9      return 0;
10 }
```



Sumário

3 Interrupção



Interrupção

- Existem dois comandos que interrompem o fluxo de execução do laço, mas de maneira diferente:
 - ▶ `break` e `continue`.
- Examinaremos cada um deles agora.



Sumário

- 3 Interrupção
 - Break
 - Continue



Interrupção: Break

- O comando `break` interrompe o fluxo do laço de repetição, fazendo com que o fluxo do programa continue exatamente após o laço.
- Útil para economizar computação, uma vez que uma determinada condição seja atingida.



Interrupção: Break

- Por exemplo: vamos tomar o problema de calcular a **menor** potência de dois que é **maior ou igual** do que um determinado inteiro x , lido do teclado. Estamos assumindo que x é sempre positivo.
- Se $x = 5$ a resposta seria 8.
- Se $x = 16$ a resposta é 16.
- Se $x = 600$, a resposta é 1024



Interrupção: Break

- A ideia aqui é abortar o laço assim que a potência seja encontrada.
- Iremos manter uma variável `pot` que iniciará de 1, a primeira potência inteira de dois, e computará a próxima potência de dois a cada iteração.
- Assim que `pot` seja maior ou igual a x , o laço será abortado.



Interrupção: Break

```
1  #include <stdio.h>
2
3  int main(void) {
4      int pot, x;
5      printf("Digite um inteiro positivo: ");
6      scanf("%d", &x);
7      pot = 1;
8      while (1) {
9          if(pot>=x)
10             break;
11         pot *= 2;
12     }
13     printf("Resposta: %d\n",pot);
14     return 0;
15 }
```



Interrupção: Break

- Equivalentemente, podemos inserir a condição do `break` no laço de repetição de modo a eliminá-lo.
- Em códigos mais complexos, quando temos múltiplas condições de parada, o uso do comando `break` pode simplificar a solução.



Interrupção: Break

```
1  #include <stdio.h>
2
3  int main(void) {
4      int pot, x;
5      printf("Digite um inteiro positivo: ");
6      scanf("%d", &x);
7      pot = 1;
8      while (pot < x)
9          pot *= 2;
10     printf("Resposta: %d\n", pot);
11     return 0;
12 }
```



Interrupção: Break

```
1  #include <stdio.h>
2
3  int main(void) {
4      int pot, x;
5      printf("Digite um inteiro positivo: ");
6      scanf("%d", &x);
7      for (pot = 1; pot < x; pot *= 2)
8          ;
9      printf("Resposta: %d\n", pot);
10     return 0;
11 }
```



Sumário

- 3 Interrupção
 - Break
 - Continue



Interrupção: Continue

- O comando `continue`, em vez de sair do laço de repetição como o `break`, ignora restante do bloco de comandos que está abaixo dele e vai direto para o teste da condição.
- Em termos práticos: pula para a próxima iteração do laço, caso haja uma próxima iteração.



Interrupção: Continue

- Por exemplo: vamos tomar o problema de imprimir todos os números de 1 a 10, exceto o 5.



Interrupção: Break

```
1  #include <stdio.h>
2
3  int main(void) {
4      int i;
5      for(i=1;i<=10;i++){
6          if(i==5){
7              continue;
8          }
9          printf("%d ",i);
10     }
11     printf("\n");
12     return 0;
13 }
```



Interrupção: Continue

- Por exemplo: vamos tomar o problema de imprimir a área de todos os círculos com raio entre 1 e 10.
- Contudo, caso o raio seja múltiplo de 4 a área não deve ser impressa.



Interrupção: Break

```
1  #include <stdio.h>
2
3  int main(void) {
4      int r;
5      const double pi = 3.1415;
6
7      for (r = 1; r <= 10; r++) {
8          if (r % 4 == 0)
9              continue;
10         double area = pi * r * r;
11         printf("Área do círculo de raio %d = %.2f\n", r, area);
12     }
13     return 0;
14 }
```



Sumário

4 Exemplos



Sumário

4 Exemplos

- Menu
- Números Primos
- Fatorial
- Fibonacci
- Binário para decimal



Menu

- Suponha que queiramos construir uma calculadora com as quatro operações básicas.
- A ideia é apresentar um menu pro usuário com as opções de operação e a opção de sair.
- O usuário então escolhe a opção e, caso ela seja válida e não seja a de encerrar o programa, digita os dois números a serem operados, os quais devem ser lidos pelo programa e operados por ele.
- O programa então volta a apresentar as opções para o usuário e tudo se repete, até que ele opte por digitar a opção que finaliza o programa.



Menu

- Neste caso, é mais natural a escolha da estrutura `do while`, já que o menu deve ser apresentado de qualquer forma, isto é, pelo menos uma iteração do laço deve ser executada.



Menu

```
1  #include <stdio.h>
2
3  int main(void) {
4      double op1, op2;
5      int opcao;
6      do {
7          printf("1. Soma\n");
8          printf("2. Subtração\n");
9          printf("3. Multiplicação\n");
10         printf("4. Divisão\n");
11         printf("5. Sair\n");
12         printf("Digite uma das opções (1 a 5): ");
13         scanf("%d", &opcao);
14         if (opcao >= 1 && opcao <= 4) {
15             printf("Digite o primeiro número: ");
16             scanf("%lf", &op1);
17             printf("Digite o segundo número: ");
18             scanf("%lf", &op2);
19         }
```



Menu

```
20     if (opcao == 1)
21         printf("%.2f + %.2f = %.2f\n\n", op1, op2, op1 + op2);
22     else if (opcao == 2)
23         printf("%.2f - %.2f = %.2f\n\n", op1, op2, op1 - op2);
24     else if (opcao == 3)
25         printf("%.2f * %.2f = %.2f\n\n", op1, op2, op1 * op2);
26     else if (opcao == 4)
27         printf("%.2f / %.2f = %.2f\n\n", op1, op2, op1 / op2);
28     else if (opcao != 5)
29         printf("Opção inválida, digite números de 1 a 5\n\n");
30 } while (opcao != 5);
31 return 0;
32 }
```



Sumário

4 Exemplos

- Menu
- **Números Primos**
- Fatorial
- Fibonacci
- Binário para decimal



Números Primos: Verificação

- Um número primo é um número natural que possui **exatamente** dois divisores naturais.
- O número 1 não é primo, pois só possui apenas um divisor: ele mesmo.
- Em outras palavras: se o número é maior que um ele é primo se ele é divisível apenas por ele mesmo e por 1.
- Como criar um programa que verifica se um número é primo?



Números Primos: Verificação

Estratégia 1

- Dado um número x , podemos dividi-lo por todos os números no intervalo $[2, x - 1]$.
- Caso ele seja divisível por algum número, então x não pode ser primo.
- Caso x seja maior que 1 e não divisível pelos números entre $[2, x - 1]$, então x é primo.
- Exemplo: para o número 5, testaríamos se ele é divisível pelos números 2, 3 e 4.



Números Primos: Verificação

```
1  #include <stdio.h>
2
3  int main(void) {
4      int x, i;
5      int primo;
6      printf("Digite um número natural: ");
7      scanf("%d", &x);
8      primo = x != 1 ? 1 : 0;
9      for (i = 2; i < x; i++) {
10         if (x % i == 0)
11             primo = 0;
12     }
13     if (primo)
14         printf("%d é primo.\n",x);
15     else
16         printf("%d não é primo.\n",x);
17     return 0;
18 }
```



Números Primos: Verificação

- Note que estamos fazendo o uso de uma **variável indicadora** `primo` que indica se o número é ou não é primo.
- Inicialmente ela é verdadeira se o número é maior que 1 e falsa, caso contrário.
- Durante o laço, caso encontremos um número que divida x , o valor falso é atribuído à ela.
- Ao final do processamento, temos que ela é verdadeira se x é primo e falsa quando x não é primo.



Números Primos: Verificação

- Podemos fazer uma pequena otimização: assim que detectamos que o número não é primo, não há razão para continuar o processamento.
- Utilizamos o valor da variável `primo` como critério de parada!
- Para $x = 9$, testaremos os divisores 2 e 3 apenas, uma vez que 9 é divisível por 3. Não é necessário testar todos os divisores.



Números Primos: Verificação

```
1  #include <stdio.h>
2
3  int main(void) {
4      int x, i;
5      int primo;
6      printf("Digite um número natural: ");
7      scanf("%d", &x);
8      primo = x != 1 ? 1 : 0;
9      for (i = 2; i < x && primo; i++) {
10         if (x % i == 0)
11             primo = 0;
12     }
13     if (primo)
14         printf("%d é primo.\n", x);
15     else
16         printf("%d não é primo.\n", x);
17     return 0;
18 }
```



Números Primos: Verificação

```
1  #include <stdio.h>
2
3  int main(void) {
4      int x, i;
5      int primo;
6      printf("Digite um número natural: ");
7      scanf("%d", &x);
8      primo = x != 1 ? 1 : 0;
9      for (i = 2; i < x; i++) {
10         if (x % i == 0){
11             primo = 0;
12             break;
13         }
14     }
15     if (primo)
16         printf("%d é primo.\n", x);
17     else
18         printf("%d não é primo.\n", x);
19     return 0;
20 }
```



Números Primos: Verificação

- A utilização de uma **variável indicadora** é muito útil em algumas soluções.
- Ela indica se temos (1) ou não temos (0) uma determinada propriedade.
- Pode ser usada para agilizar as computações quando empregada como critério de parada.



Números Primos: Verificação

- Os programas apresentados funcionam corretamente.
- Contudo, podemos implementar um código mais eficiente.
- Se um número x não é primo, então ele pode ser escrito como um produto de dois números p e q , com $1 \leq p < x$ e $1 \leq q \leq x$.
- $x = p \cdot q$.
- Observação: p e q não podem ser, **simultaneamente**, maior que \sqrt{x} .
- Conclusão: só precisamos testar os divisores até \sqrt{x} .



Números Primos: Verificação

Estratégia 2

- Dado um número x , podemos dividi-lo por todos os números no intervalo $[2, \sqrt{x}]$.
- Caso ele seja divisível por algum número, então x não pode ser primo.
- Caso x seja maior que 1 e não divisível pelos números entre $[2, \sqrt{x}]$, então x é primo.
- Exemplo: para o número 17, testaríamos se ele é divisível pelos números 2, 3 e 4, haja vista que $\sqrt{17} \approx 4.12$.



Números Primos: Verificação

```
1  #include <math.h>
2  #include <stdio.h>
3  int main(void) {
4      int x, i;
5      int primo;
6      printf("Digite um número natural: ");
7      scanf("%d", &x);
8      primo = x != 1 ? 1 : 0;
9      for (i = 2; i <= sqrt(x) && primo; i++) {
10         if (x % i == 0)
11             primo = 0;
12     }
13     if (primo)
14         printf("%d é primo.\n", x);
15     else
16         printf("%d não é primo.\n", x);
17     return 0;
18 }
```



Sumário

4 Exemplos

- Menu
- Números Primos
- **Fatorial**
- Fibonacci
- Binário para decimal



Fatorial

- O fatorial de um número inteiro $n \geq 0$ é definido da seguinte forma:

$$n! = \begin{cases} 1, & n = 0 \\ 1, & n = 1 \\ n \cdot (n - 1) \cdot \dots \cdot 1, & n > 1 \end{cases}$$

- $0! = 1$.
- $1! = 1$
- $5! = 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 = 120$.
- Vamos escrever um programa que calcule $n!$, dado um n lido do usuário.



Fatorial

Estratégia

- Uma estratégia para resolver este problema é utilizar uma **variável acumuladora** `fat`.
- Esta variável armazena os valores intermediários da computação do fatorial a cada iteração.
- Ao final da computação, a variável conterá o resultado definitivo.
- Na i -ésima iteração, a variável `fat` conterá o resultado de $i!$.



Fatorial

```
1  #include <stdio.h>
2
3  int main(void) {
4      int i;
5      int fat;
6      int n;
7      printf("Digite o valor de n: ");
8      scanf("%d", &n);
9      for (i = 2, fat = 1; i <= n; i++) {
10         fat *= i;
11     }
12     printf("%d! = %d\n",n,fat);
13     return 0;
14 }
```



Fatorial

- A utilização de uma **variável acumuladora** está presente nas soluções de muitos problemas.
- Podemos empregá-la sempre que o resultado de uma iteração depende imediatamente da anterior.



Sumário

4 Exemplos

- Menu
- Números Primos
- Fatorial
- **Fibonacci**
- Binário para decimal



Fibonacci

- O n -ésimo item da sequência de Fibonacci pode ser definido através da seguinte equação:

$$F(n) = \begin{cases} 1, & n = 0 \\ 1, & n = 1 \\ F(n-1) + F(n-2), & n > 1 \end{cases} \quad (1)$$

- Assim, a sequência é constituída dos números: 1, 1, 2, 3, 5, 8, 13, ...
- Cada número é igual a soma dos dois anteriores, com exceção dos dois primeiros, que valem 1.



Fibonacci

- Problema: dado um inteiro $n \geq 0$, computar $F(n)$.



Fibonacci

Estratégia

- Podemos resolver este problema mantendo duas variáveis acumuladoras, `fib_1` e `fib_2`.
- A cada iteração estas variáveis são atualizadas para armazenar o último e o penúltimo número de Fibonacci vistos.
- Com a posse delas, é possível calcular o próximo número de Fibonacci.



Fibonacci

```
1  #include <stdio.h>
2
3  int main(void) {
4      int i, n;
5      int fib_1, fib_2;
6      int resposta = 1;
7      printf("Digite o valor de n: ");
8      scanf("%d", &n);
9      for (i = 2, fib_1 = 1, fib_2 = 1; i <= n; i++) {
10         resposta = fib_1 + fib_2;
11         int aux = resposta;
12         fib_2 = fib_1;
13         fib_1 = aux;
14     }
15     printf("F(%d) = %d.\n", n, resposta);
16     return 0;
17 }
```



Sumário

4 Exemplos

- Menu
- Números Primos
- Fatorial
- Fibonacci
- Binário para decimal



Binário para Decimal

- Problema: dado um número inteiro, contendo apenas dígitos 0 e 1, representando um número binário, convertê-lo para decimal.
- Exemplo: $11001 = 25$.



Binário para Decimal

- Como extrair cada dígito?
- Podemos dividir pela potência de 10 adequada e tirar o resto por 10:

▶ **1**956: `1956/1000 % 10 == 1` .

▶ **19**56: `1956/100 % 10 == 9` .

▶ **195**6: `1956/10 % 10 == 5` .

▶ **1956**: `1956/1 % 10 == 6` .



Binário para Decimal

Estratégia

- Podemos utilizar duas variáveis acumuladoras:
 - ▶ `pot_2` : armazena o valor da próxima potência de dois.
 - ▶ `soma` : armazena o valor da soma de todas as potências de dois cujo bit 1 está ligado.
- A cada iteração `soma` recebe o valor do bit vezes o valor `pot_2`.



Binário para Decimal

```
1  #include <stdio.h>
2
3  int main(void) {
4      int n;
5      int n_original;
6      int pot_2 = 1, soma = 0;
7      printf("Digite um número em binário: ");
8      scanf("%d", &n);
9      n_original = n;
10     do {
11         int bit = n % 10; // obtém o valor do dígito menos significativo
12         soma += bit * pot_2;
13         n /= 10; // divide o número por 10 a cada iteração
14         pot_2 *= 2; // obtém a próxima potência de dois
15     } while (n > 0);
16     printf("%d = %d\n", n_original, soma);
17     return 0;
18 }
```



Sumário

5 Laços Aninhados



Laços Aninhados

- Em muitos casos, para resolver um problema, é necessário empregar uma solução que envolve laços de repetição dentro de laços de repetição.
- **Laços Aninhados.**



Laços Aninhados

- Problema: imprimir todos os pares de números, que devem estar entre 1 e 100, que somam um inteiro c , informado pelo usuário.



Laços Aninhados

Estratégia

- A ideia é utilizar dois laços aninhados.
- O primeiro laço fixa um valor i , enquanto o segundo, interno, varia o valor j de 1 a 100.
- Toda vez que o laço interno finaliza, incrementamos o valor de i para que se possa reiniciar o processo com j novamente variando de 1 a 100.



Laços Aninhados: Soma de Pares

```
1  #include <stdio.h>
2
3  int main(void){
4      int c;
5      int i,j;
6      printf("Digite o valor de c: ");
7      scanf("%d",&c);
8      for(i=1;i<=100;i++){
9          for(j=1;j<=100;j++){
10             if(i+j == c){
11                 printf("%d + %d = %d\n",i,j,c);
12             }
13         }
14     }
15     return 0;
16 }
```



Laços Aninhados

- E se quiséssemos imprimir todos os pares **distintos** com tal propriedade?
- Basta fazer com que `j=i` na inicialização do laço interno.



Laços Aninhados: Soma de Pares

```
1  #include <stdio.h>
2
3  int main(void){
4      int c;
5      int i,j;
6      printf("Digite o valor de c: ");
7      scanf("%d",&c);
8      for(i=1;i<=100;i++){
9          for(j=i;j<=100;j++){
10             if(i+j == c){
11                 printf("%d + %d = %d\n",i,j,c);
12             }
13         }
14     }
15     return 0;
16 }
```



Laços Aninhados

- Veremos agora uma série de problemas em que podemos aplicar laços aninhados em suas soluções.



Sumário

- 5 Laços Aninhados
 - Dados
 - Números Primos
 - Interrupção em laços aninhados



Laços Aninhados: Dados

- Problema: imprimir todas as possíveis combinações do lançamento de 6 dados com 6 faces cada um.



Laços Aninhados: Dados

Estratégia

- 6 laços aninhados.
- Cada laço ficará responsável pelo valor de um determinado dado.



Laços Aninhados: Dados

```
1  #include <stdio.h>
2
3  int main(void) {
4      int dado_1, dado_2, dado_3, dado_4, dado_5, dado_6;
5      for (dado_1 = 1; dado_1 <= 6; dado_1++) {
6          for (dado_2 = 1; dado_2 <= 6; dado_2++) {
7              for (dado_3 = 1; dado_3 <= 6; dado_3++) {
8                  for (dado_4 = 1; dado_4 <= 6; dado_4++) {
9                      for (dado_5 = 1; dado_5 <= 6; dado_5++) {
10                         for (dado_6 = 1; dado_6 <= 6; dado_6++) {
11                             printf("Lançamento: %d %d %d %d %d %d\n", dado_1,
12                                 dado_2, dado_3, dado_4, dado_5, dado_6);
13                         }
14                     }
15                 }
16             }
17         }
18     }
19     return 0;
20 }
```



Sumário

- 5 Laços Aninhados
 - Dados
 - Números Primos
 - Interrupção em laços aninhados



Laços Aninhados: Números Primos

- Problema: dado um inteiro $n \geq 0$, imprimir os n primeiros números primos.



Laços Aninhados: Números Primos

Estratégia

- Já sabemos detectar se um número é ou não é primo.
- A estratégia aqui é manter uma variável contadora `cnt` e atualizá-la toda vez que um primo for encontrado até que ela chegue ao valor n .
- Devemos checar se a variável `candidato` contém ou não um número primo. Ela começa de 2 e é incrementada antes de cada verificação de número primo.



Laços Aninhados: Números Primos

```
1  #include <math.h>
2  #include <stdio.h>
3
4  int main(void) {
5      int i, cnt, n, candidato, primo;
6      printf("Informe a quantidade de números primos a serem impressos: ");
7      scanf("%d", &n);
8      for (cnt = 0, candidato = 2; cnt < n; candidato++) {
9          primo = 1;
10         for (i = 2; i <= sqrt(candidato) && primo; i++) {
11             if (candidato % i == 0)
12                 primo = 0;
13         }
14         if (primo) {
15             printf("%d ", candidato);
16             cnt++;
17         }
18     }
19     printf("\n");
20     return 0;
21 }
```



Sumário

- 5 Laços Aninhados
 - Dados
 - Números Primos
 - Interrupção em laços aninhados



Interrupção em laços aninhados

- É importante ressaltar que os comandos `break` e `continue` aplicam-se apenas ao laço em que foi aplicado.
- Deve-se ter cuidado de onde colocá-los para obtenção do comportamento desejado.



Laços Aninhados: Números Primos

```
1  #include <math.h>
2  #include <stdio.h>
3  int main(void) {
4      int i, cnt, n, candidato, primo;
5      printf("Informe a quantidade de números primos a serem impressos: ");
6      scanf("%d", &n);
7      for (cnt = 0, candidato = 2; cnt < n; candidato++) {
8          primo = 1;
9          for (i = 2; i <= sqrt(candidato); i++) {
10             if (candidato % i == 0) {
11                 primo = 0;
12                 break;
13             }
14         }
15         if (primo) {
16             printf("%d ", candidato);
17             cnt++;
18         }
19     }
20     printf("\n");
21     return 0;
22 }
```