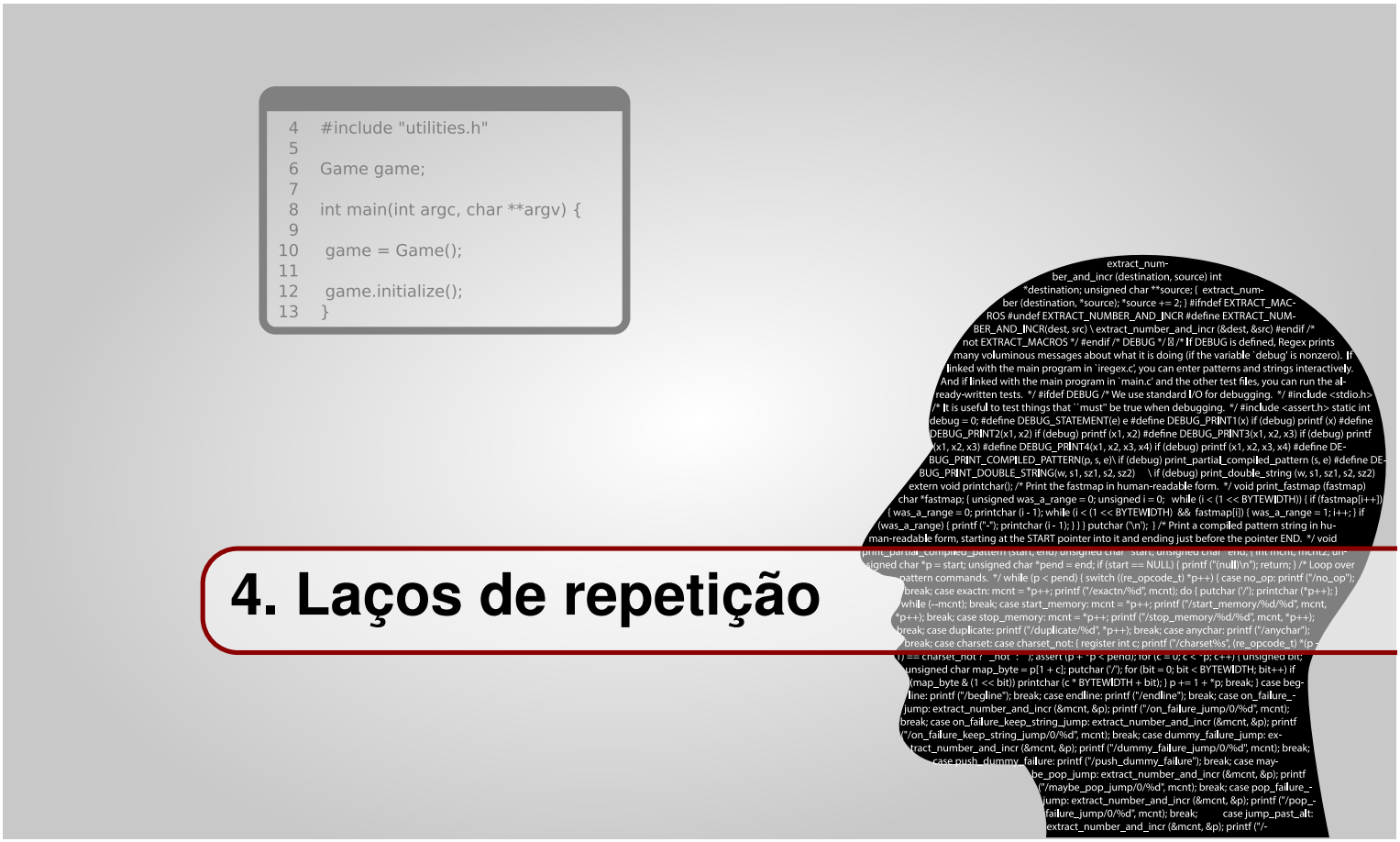


# 4. Laços de repetição

No capítulo anterior, vimos que para evitar a reprodução repetida de códigos, uma das maneiras em linguagens imperativas consiste no uso de estruturas (ou laços) de repetição. Este capítulo trata dos laços de repetição utilizados na linguagem C e certos aspectos de alguns problemas particulares.

## 4.1 Sintaxe das estruturas de repetição na linguagem C

A primeira estrutura que se apresenta é o **while**, cuja sintaxe está exposta na Listagem 4.1. A leitura lógica desta estrutura é "*enquanto a expr_lógica for verdadeira, repita o bloco de códigos*". Neste sentido, podemos concluir que a expressão é avaliada antes mesmo de realizar a primeira repetição.

Listing 4.1: Estrutura do laço de repetição while.

```
1 while(<expr_lógica>){
2   Bloco_de_repetição
3 }
```

■ **Exemplo 4.1** Suponha que queiramos repetir a frase "Eu vou passar em LP!"por 10 vezes.

Portanto, um código em C usando o laço de repetição while que resolve o problema acima é exemplificado na Listagem 4.2.

Listing 4.2: Exemplo utilizando o laço de repetição while.

```
1 #include<stdio.h>
2 int main(){
3   int cont=1;
4   while(cont <= 10){
```

```
5     puts("Eu vou passar em LP!");
6     cont++;
7   }
8   return 0;
9 }
```

A outra estrutura de repetição é o comando do-while. A interpretação lógica deste comando é: "*Execute o bloco de códigos depois verifique se efetua a repetição (caso a expressão lógica seja verdadeira)*". Diferente do while apenas, o do-while permite pelo menos uma execução do bloco de códigos que se encontra dentro da estrutura de repetição. Depois que é executado, a estrutura avalia se a expressão lógica é verdadeira, caso seja, volta e executa novamente o bloco de códigos.

Listing 4.3: Estrutura do laço de repetição do-while.

```
1 do{
2   Bloco_de_repetição
3 }while(<expr_lógica>);
```

■ **Exemplo 4.2** Considere o seguinte problema: leia um número indefinido de valores inteiros, sendo que o último número lido e processado deve ser 0 (zero). Para cada valor lido $x$, o programa deverá mostrar o valor de $x^2$.

Listing 4.4: Exemplo utilizando o laço de repetição do-while.

```
1  #include<stdio.h>
2  int main(){
3    int x;
4    do{
5      puts("Digite o número");
6      scanf("%d", &x);
7      printf("%d\n", x*x);
8    }while(x!=0);
9    return 0;
10 }
```

A terceira estrutura de repetição é o for. Este tem uma estrutura peculiar que se divide em até três partes. A estrutura sintática deste laço encontra-se na Listagem 4.5.

Listing 4.5: Estrutura do laço de repetição for.

```
1 for(<init>; <expr_lógica>; <pos>){
2   Bloco_de_repetição
3 }
```

As partes do for podem ser explicadas da seguinte maneira:

1. `<init>`: este bloco é executado apenas uma vez e antes de ser verificada a validade da expressão lógica. Geralmente, coloca-se aqui a inicialização de contadores.

2. <expr_lógica>: esta é a expressão lógica que valida se o bloco de códigos vai ser executado. Ela é verificada de maneira semelhante ao laço while, ou seja, antes de executar o bloco, a verificação da veracidade da expressão lógica é efetuada.
3. <pos>: esta parte é executada de forma posterior ao bloco de códigos. Ou seja, após a última instrução executada no bloco de códigos, o que estiver em <pos> será executado logo em seguida. Geralmente se coloca o incremento de variáveis contadores nesta parte.

■ **Exemplo 4.3** Suponha que queiramos repetir a frase "Eu vou passar em LP!"por 10 vezes.

Portanto, um código em C usando o laço de repetição for que resolve o problema acima é exemplificado na Listagem 4.6.

Listing 4.6: Exemplo utilizando o laço de repetição while.

```
1  #include<stdio.h>
2  int main(){
3  int i;
4  for(i=1; i <= 10; i++)
5    puts("Eu vou passar em LP!");
6  return 0;
7  }
```

### Lista 4.1 - Exercícios propostos

**Problema 4.1** Crie um programa em C que receba cinco números e imprima o cubo de cada número lido.

```
1  #include<stdio.h>
2  int main(){
3    int num, cont=0;
4    while(cont++<5){
5      printf("Entre com o numero %d:", cont);
6      scanf("%d", &num);
7      printf("O cubo de %d eh %d\n", num, num*num*num);
8    }
9    return 0;
10 }
```

**Problema 4.2** Faça um programa em C que imprima todos os números pares no intervalo de 1 a 10.

```
1  #include<stdio.h>
2  int main(){
3    int par;
4    for(par=2; par<=10; par+=2)
5      printf("%d, ", par);
6    return 0;
7  }
```

**Problema 4.3** Faça um programa em C para ler 10 números flutuantes e imprimir a metade de cada número.

**Problema 4.4** Faça um programa que calcule e imprima o valor de $b^n$. O valor de $n$ deverá ser maior que 1 e inteiro e o valor de $b$ maior ou igual a 2 e inteiro.

**Problema 4.5** Faça um programa que imprima uma tabela com a conversão de centímetros para polegadas. A tabela deve constar os primeiros 20 centímetros (inteiros). Use a seguinte equivalência: 1polegada=2,54cm.

**Problema 4.6** Faça um programa que leia dois valores inteiros positivos ($a$ e $b$). Em seguida, mostre os números inteiros pares contidos no intervalo $(a, b]$. O programa deve certificar que $a > b$.

**Problema 4.7** Faça um programa que leia um número $n$ inteiro positivo e imprima os $n$ primeiros números positivos múltiplos de 5.

## 4.2 Acumuladores

Suponha o seguinte problema: faça um algoritmo que leia $n$ números inteiros e mostre a soma dos números lidos. Para resolver problemas que é preciso fazer somatórios ou produtórios, precisamos de variáveis acumuladoras. No caso do problema acima, poderíamos colocar em termos mate´máticos que seria: $\sum_i^n num_i$, sendo $num_i$ o $i$-ésimo número lido. Portanto uma solução poderia ser:

```c
#include<stdio.h>
int main(){
  int soma=0, num;
  unsigned int n, i;
  puts("Entre com a quantidade de números:");
  scanf("%u", &n);
  for(i=1; i<=n; i++){
    printf("Entre com o item %u de um total de %u", i, n);
    scanf("%d", &num);
    soma +=num;
  }
  printf("A soma dos itens digitados é: %d\n", soma);
  return 0;
}
```

Repare que se fosse pedido o produtório $\prod_{i=1}^n num_i$, repare que nossa variável acumuladora teria que ser inicializada com 1 (um), em lugar de 0 (zero).

```c
#include<stdio.h>
int main(){
  int prod=1, num;
  unsigned int n, i;
  puts("Entre com a quantidade de números:");
  scanf("%u", &n);
  for(i=1; i<=n; i++){
    printf("Entre com o item %u de um total de %u", i, n);
    scanf("%d", &num);
```

```
10      prod *=num;
11    }
12    printf("O produto dos itens digitados é: %d\n", prod);
13    return 0;
14  }
```

■ **Exemplo 4.4** O fatorial de um número $n$ é dado pela seguinte fórmula: $n! = n * (n-1) * ... * 2 * 1$. Faça um programa em C que leia um número e calcule seu fatorial.

```
1  #include<stdio.h>
2  int main(){
3    unsigned int n, fat, i;
4    fat=1;
5    puts("Entre com o numero: ");
6    scanf("%u%, &n);
7    for(i=2; i<=n; i++)
8      fat*=i;
9    printf("O fatorial eh: %u\n", fat);
10   return 0;
11 }
```

De posse deste conhecimento, resolva os seguintes problemas:

**Lista 4.2 - Acumuladores**

**Problema 4.8** Faça um programa que leia um número $n$ inteiro positivo e imprima o produtório dos múltiplos de 3 no intervalo de $[1, n]$.

**Problema 4.9** Faça um programa que leia um número $n$ inteiro positivo e imprima a **soma** dos $n$ primeiros números positivos múltiplos de 5.

**Problema 4.10** Faça um programa que leia dois números $a$ e $b$, inteiros, positivos. Se $a < b$, calcule e imprima a soma dos números inteiros no intervalo (a, b).

**Problema 4.11** Faça um programa que leia três números $a$, $b$ e $n$, inteiros e positivos. Se $a < b$, calcule e imprima o produto dos números inteiros divisíveis por $n$ no intervalo [a, b].

## 4.3 As instruções break e continue

O que aconteceria se eu precisasse encerrar ou continuar o meu laço de repetição antes de executar parte do código? Estas duas situações são previstas na linguagem C através dos comandos **break** e **continue**.

O comando break permite a quebra do laço de repetição de onde for chamado. Enquanto que o comando continue permite que o laço retorne ao seu início. Vejamos o uso destas estruturas no exemplo abaixo.

■ **Exemplo 4.5** Suponha que o problema a ser resolvido seja este: leia $n$ números inteiros positivos pares, calcule e mostre a soma deles. O programa deverá se certificar que o número lido é positivo e par.

```c
#include<stdio.h>
int main(){
  unsigned int n, d, cont=0, soma=0;
  puts("Entre com a quantidade de números:");
  scanf("%u", &n);
  while(cont<n){
    puts("Entre com o num %u:", cont+1);
    scanf("%u", &d);
    if( d % 2 != 0)
      continue;
    soma+= d;
    cont++;
  }
  printf("Soma=%u\n", soma);
  return 0;
}
```

■ **Exemplo 4.6** Um número $n$ é dito ser primo se tem apenas dois divisores: 1 e $n$. Portanto, para verificar se um número é primo, deve-se certificar que não há divisores inteiros no intervalo $(1, n)$. Sendo assim, um código para realizar esta tarefa poderia ser:

```c
#include<stdio.h>
int main(){
  unsigned int n, div, primo=1;
  puts("Entre com o número n: ");
  scanf("%u", &n);

  for(div=2; div<n; div++)
    if(n % div == 0){
      primo=0;
      break;
    }
  if(primo)
    puts("É primo!");
  else
    puts("Não é primo");
  return 0;
}
```

### 4.3.1 Séries

Outro aspecto bastante comum envolvendo laços de repetição são as séries. Uma série famosa na matemática e computação é a série de Fibonacci. A série de Fibonacci consiste na seguinte sequência $0, 1, 1, 2, 3, 5, 8, 13, \ldots$

Se você prestou bem atenção, já pode perceber que definidos os dois primeiros

números 0, 1, os demais são obtidos somando-se os dois anteriores. Esta noção fica melhor representada, para $n > 0$, na seguinte equação de recorrência

$$T_n = \begin{cases} n & \text{se } n < 2 \\ T_{n-1} + T_{n-2} & \text{caso contrário} \end{cases} \tag{4.1}$$

De posse da definição, veja o seguinte problema.

**Problema 4.12** Faça um programa para ler um número inteiro $n > 0$ e imprima a sequência de Fibonacci, $T_1, ..., T_n$,.

### 4.3.2 Laços aninhados

Até o momento vimos exemplos e problemas que necessitavam de apenas um laço de repetição. Sobretudo, nada nos impede de colocar um laço dentro de outro laço de repetição. Vamos proceder com esta possibilidade através de exemplos.

■ **Exemplo 4.7** Faça um programa em linguagem C para mostrar os números primos no intervalo (0,1000). Poderíamos desmontar o problema na seguinte lógica: faço um laço com uma variável contadora ($n$) que começa em 1 e vai até 999. Para cada valor de $n$, usamos o código que verifica se é primo ou não. Caso seja primo, imprimimos $n$.

```
1  #include<stdio.h>
2  int main(){
3  unsigned int n, div, primo;
4
5  for(n=1, n<1000; n++){
6    for(primo=1, div=2; div<n; div++)
7      if(n % div == 0){
8        primo=0;
9        break;
10     }
11
12   if(primo)
13     printf("%d, ", n);
14 }
15   return 0;
16 }
```

■ **Exemplo 4.8** Considere o seguinte problema:

**Problema 4.13** Faça um algoritmo que leia uma quantidade $n$ de linhas, e para cada linha $i \in 1, 2, \ldots, n$ imprima o número 1, $i$ vezes.

```
1  #include<stdio.h>
2  int main(){
3    unsigned int n, linha, conta1;
4    puts("Digite a quantidade de linhas: ");
5    scanf("%d", &n);
```

```
6   for(linha=1; linha <= n; linha++){
7     for(conta1=0; conta1<linha; conta1++)
8       printf("1 ");
9     printf("\n");
10  }
11  return 0;
12 }
```

Na tentativa de deixar mais claro que o laço de repetição mais interno "roda mais rápido"que o externo, considere o seguinte exemplo:

■ **Exemplo 4.9**

**Problema 4.14** Gere os números de três dígitos, onde cada um dos dígitos é um número ímpar.

Uma alternativa seria a de criar um contador que começasse em 111 e fosse até 999. Dai você desmonta os dígitos e verifica se todos são números ímpares. Porém, a alternativa abaixo me parece mais rápida e adequada para resolver o problema.

```
1  #include<stdio.h>
2  int main(){
3    int d1, d2, d3;
4    for(d1=1; d1<10; d1+=2)
5      for(d2=1; d2<10; d2+=2)
6        for(d3=1; d3<10; d3+=2)
7          printf("%d%d%d\n", d1, d2, d3);
8    return 0;
9  }
```

## 4.4 Loops (teoricamente) infinitos

Ler um determinado número de entradas, até que as entradas sejam 0.

```
1  #include<stdio.h>
2  int main(){
3    int n;
4    while(1){
5      puts("Entre com o n:");
6      scanf("%d", &n);
7      if(n==0) break;
8      printf("O quadrado de %d é %d\n", n, n*n);
9    }
10   return 0;
11 }
```

### Lista 4.3 - Laços de Repetição - Exercícios propostos

**Problema 4.15** Faça um algoritmo que leia um número N de notas, em seguida leia as N notas, calcule e mostre a média aritmética das N notas. Garanta que N é

um número maior que 0.

```c
##include <stdio.h>
/**
* Loops em C!!
*/
int main() {
  int N, i;
  float nota_atual;
  float soma=0.0;
  puts("Digite o numero de notas: ");
  scanf("%d", &N);
  if(N>0){
    for(i=1; i<=N; i++){
      printf("Digite a nota numero %d: ", i);
      scanf("%f", &nota_atual);
      soma+=nota_atual;
    }
    printf("A média das notas lidas é: %f", soma/N);
  }
  return 0;
}
```

**Problema 4.16** Faça um programa que leia um número inteiro qualquer e imprima a soma de seus dígitos.

**Problema 4.17** Faça um programa que receba um número inteiro positivo N < 100, leia os N números, calcule e mostre a soma dos números pares e a soma dos números ímpares.

```c
#include <stdio.h>
/**
* Loops em C!!
*/
int main() {
  unsigned int N, i, num_atual,
  soma_pares=0, soma_impares=0;
  puts("Digite a qtde de números: ");
  scanf("%u", &N);
  if(N>=100){
    puts("Número deve ser menor que 100!");
    return 0;
  }
  for(i=1; i<=N; i++){
    printf("Digite o numero %d: ", i);
    scanf("%u", &num_atual);
    if(num_atual % 2 == 0) //Num eh par
      soma_pares+=num_atual;
    else
```

```
20        soma_impares+= num_atual;
21    }
22    printf("Soma pares: %d Soma ímpares: %d\n", soma_pares,
          soma_impares);
23    return 0;
24  }
```

**Problema 4.18** Faça um programa que receba 10 números inteiros e mostre a quantidade de números primos dentre os números que foram digitados.

**Problema 4.19** Faça um algoritmo que leia duas datas no formato dd/mm/YYYY e calcule a diferença destas datas em dias. Procure na internet como se verifica se um determinado ano é bissexto.

**Problema 4.20** Uma empresa fez uma pesquisa de mercado para saber se as pessoas gostaram ou não de um novo produto lançado. Para isso, forneceu o sexo do entrevistado e sua resposta (S - Sim; ou d - não). Sabe-se que foram entrevistadas dez pessoas. Faça um programa que calcule e mostre:

1. O número de pessoas que responderam sim;
2. o número de pessoas que responderam não;
3. o número de mulheres que responderam sim; e
4. a percentagem de homens que responderam não, entre todos os homens analisados.

### Lista 4.4 - Laços de Repetição

**Problema 4.21** Primos quadrados perfeitos Diz-se que dois números inteiros positivos $N_1$ e $N_2$ são ditos primos quadrados perfeitos se, a soma dos dígitos de $(N_1)^2$ é igual a $N_2$ e a soma dos dígitos de $(N_2)^2$ é igual a $N_1$. Por exemplo, os números 16 e 13 são primos quadrados perfeitos, pois $16^2 = 256$, e $2 + 5 + 6 = 13$ e $13^2 = 169$, pois $1 + 6 + 9 = 16$. Faça um algoritmo que leia uma sequência de pares de números (até que um dos números lidos seja igual a 0) e verifique se os números lidos são primos quadrados perfeitos.

| **Exemplo de Entrada** | **Exemplo de Saída** |
|---|---|
| 29 9 | 0 |
| 13 16 | 1 |
| 0 0 | |

**Problema 4.22** (Enviada por Tiago Almeida) Uma máquina de jogos de dados possui dois dados $d_1(t)$ e $d_2(t)$ que o usuário pode lançar, e cada jogada fica registrada na variável $t$. Por exemplo, $d_1(1)$ e $d_2(1)$ indica o primeiro lançamento dos dados. Na primeira jogada dos dados, ou seja $jog_1 := d_1(1) + d_2(1)$, caso $jog_1 = 11$ ou $jog_1 = 7$, o jogador ganha. Sobretudo, caso não ganhe no primeiro lançamento, o jogador pode realizar mais lançamentos. Neste caso, em qualquer lançamento $t > 1$, caso $jog_t = jog_1$, o jogador ganha. Entretanto, caso a $jog_t = 11$ ou $jog_t = 7$, então o jogador perde. Faça um programa em C que simule este jogo, sempre mostrando a soma dos dados gerados aleatoriamente a cada jogada.

**Problema 4.23** (Felipe Barbosa) As crianças são ensinadas a adicionar vários dígitos da direita para a esquerda, um dígito de cada vez. Muitos acham a operação "vai 1"(em inglês chamada de "*carry*", na qual o valor 1 é carregado de uma posição

para ser adicionado ao dígito seguinte) um desafio significativo. Seu trabalho é para contar o número de operações de *carry* para cada um dos problemas de adição apresentados para que os educadores possam avaliar a sua dificuldade.

**Entrada**

Cada linha de entrada contém dois inteiros sem sinal com no máximo 9 dígitos. A última linha de entrada contém 0 0.

**Saída**

Para cada linha de entrada, com exceção da última, você deve computar e imprimir a quantidade de operações "leva 1"que resultam da adição dos 2 números, no formato apresentado no exemplo abaixo.

| Exemplos de Entradas | Exemplos de Saídas |
|---|---|
| 123 456 | No carry operation. |
| 555 555 | 3 carry operations. |
| 123 594 | 1 carry operation. |
| 0 0 | |