Um homem sem lembranças é um homem perdido.

Armand Salacrou

1 Lembrar é viver!! Relembrando contadores e acumuladores

Na última aula, foi visto o conceito de variáveis contadoras e acumuladoras. Vamos revisar estes conceitos com outras aplicações.

1.1 Somatórios

Considere o seguinte problema:

$$\sum_{i=1}^{n} i \times (i+1)$$

De acordo com o enunciado do problema, **n** termos de uma somatória precisam ser calculados e somados. A soma é uma operação binária, isto é, ela tem dois operandos. Como podemos somar apenas dois valores a cada vez, uma estratégia possível é somarmos a cada novo termo o valor de tal termo com a soma acumulada dos termos anteriores ao do termo que acaba de ser computado. O valor resultante passa a ser o novo valor dos termos até então calculados.Ao final do processo temos, então, acumulado a soma de todos os termos.

Vamos, então, supor que o valor das somas acumuladas encontra-se armazenada em uma variável de nome *somatoria*. Além do valor das somas sucessivas de termos precisamos contabilizar o número de termos já calculados, pois queremos somar exatamente **n** termos. Para tal contabilidade usamos um contador (isto é, uma variável cujo valor deve ser incrementado em uma unidade a cada vez que um novo termo é calculado) que denominaremos de *contador* (para representar o valor **i** da equação). O valor de tal contador é usado inclusive no cálculo dos termos.

Que valores devem ser atribuídos às variáveis que representam o contador *contador* e o acumulador de somas *somatoria*? Como queremos variar o contador de 1 até **n**, um valor inicial natural para *contador* é o valor 1 (um) e, para *somatoria*, o valor 0 (zero) já que 0 (zero) representa o elemento neutro da soma e também caracteriza bem a situação inicial em que nenhum termo ainda foi calculado e acumulado.

O programa proposta pode então, ser escrito conforme o programa 1.

Programa 1: Programa para cálculo do somatório.

```
1  // Contadores e acumuladores
   // programa_001.cpp
   #include "biblaureano.h"

   int main()
6  {
     int contador, somatoria, qtdTermosN, termo;
     qtdTermosN = readInt("Entre com o valor de N:");

     contador = 1;
```

```
  somatoria = 0; //elemento neutro da soma

  while( contador <= qtdTermosN )
  {
    termo = contador * (contador+1);
    somatoria += termo; // variável somadora
    ++contador; //variável contadora
  }
  cout << "A somatória é :" << somatoria << endl;
  cout << "Game over!" << endl;

  return 0;
}
```

1.2 Produtórios

Considere o seguinte problema:

$$\prod_{i=1}^{n} i \times (i+1)$$

O problema proposto, ao invés de requerer o acúmulo de somas, requer o acúmulo de produtos. A estrutura do programa é similar à do programa desenvolvido para o problema do cálculo de uma somatória (programa 1). Ao invés de somas sucessivas, precisamos agora efetuar produtos sucessivos. O valor inicial a ser atribuído à variável que exerce o papel de acumulador agora não pode ser mais 0 (zero), mas deve ser um já que tal valor representa o elemento neutro da multiplicação. O algoritmo pode, então, ser esboçado conforme os programa 2.

Programa 2: Programa para cálculo do produtório.

```
// Contadores e produtorios
// programa_002.cpp
#include "biblaureano.h"

int main()
{
  int contador, produtoria, qtdTermosN, termo;
  qtdTermosN = readInt("Entre com o valor de N:");

  contador = 1;
  produtoria = 1; //elemento neutro da multiplicação

  while( contador <= qtdTermosN )
  {
    termo = contador * (contador+1);
    produtoria *= termo; // variável acumuladora de produtórios
    ++contador; //variável contadora
  }
  cout << "A produtória é :" << produtoria << endl;
  cout << "Game over!" << endl;
```

```
22    return 0;
}
```

2 Relembrando as formas de controle de estruturas de repetição

Lembra dos conceitos de estruturas de repetição *definidas e garantidas* e *indefinidas e sem garantia* e com controles *automáticos* ou *controlados pelo usuário* ?

Para facilitar:

- Um programa, cuja condição na estrutura de repetição, realiza repetições definidas (um cálculo, por exemplo) é dita como um programa com estrutura de repetição *automática* com fim *definido e garantido*, pois sabe-se o início e o fim do ciclo de repetições.
- Um programa, cuja condição na estrutura de repetição, realiza repetições de acordo com a resposta do usuário (uma pergunta de "Sim" ou "Não", por exemplo), é um programa *controlado pelo usuário* e com fim *indefinido e sem garantia*, pois o programa não tem controle de quantas repetições.

Vamos a exemplos das duas situações (programa 3):

Programa 3: Exemplo de repetições definidos e indefinidos.

```
// Programa com os diversos
2   // tipos de controle
// programa_003.cpp
#include "biblaureano.h"

int main()
7   {
  char resposta = ' ';

  //controlado pelo usuário
  //e sem fim definido
12   while( resposta != 's' && resposta != 'n')
  {
    resposta = readChar("Entre com s ou n:");
  }
  cout << "Game over!" << endl;
17

  int contador = 10;
  //controlado pelo programa (automático)
  //com fim definido
  while( contador > 0 )
22   {
    cout << contador << endl;
    --contador;
    espera(100); // dorme um 1 segundo
  }
27

  cout << "Game over!!" << endl;
  return 0;
}
```

3 Interrompendo uma estrutura de repetição

Às vezes é necessário quebrar a execução de um comando de repetição devido a uma condição determinada. Pode-se programar esta condição no próprio local da condição dos comandos de repetição ou colocar um teste dentro do bloco de comandos.

Caso a condição seja alcançada pode-se interromper a repetição uma maneira não usual, terminando a execução deste comando.

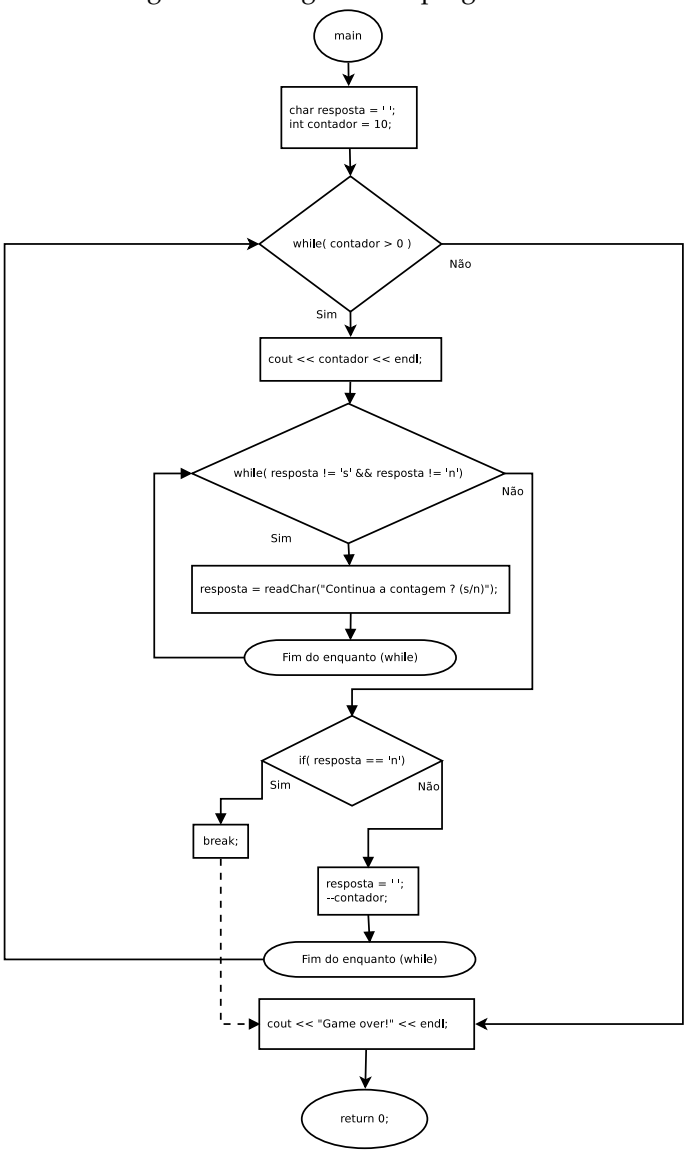
Logo, as estruturas de repetição permitem o uso do comando *break*, que causa a saída imediata do laço de repetição. Embora esta técnica esteja em desacordo com os princípios da boa programação, é um recurso que não pode ser desprezado. Observe o programa 4:

Programa 4: Uso do comando *break*.

```
// Programa com os diversos
// tipos de controle
// programa_004.cpp
#include "biblaureano.h"

int main()
{
  char resposta = ' ';
  int contador = 10;

  while( contador > 0 ) //sai quando chegar em 0
  {
    cout << contador << endl;

    while( resposta != 's' && resposta != 'n')
    {
      resposta = readChar("Continua a contagem ? (s/n)");
    }

    //interrompe a execução
    if( resposta == 'n')
    {
      break;
    }
    resposta = ' ';
    --contador;
  }
  cout << "Game over!!" << endl;
  return 0;
}
```

O programa 4 é o típico exemplo de estrutura de repetição *com fim definido* mas *controlado pelo usuário*, afinal, o usuário *decide* se o programa continua a contagem ou não. Observe o funcionamento do comando *break* na figura 1.

Figura 1: Fluxograma do programa 4.



4 Exercícios

1. Ler dois valores inteiros e imprimir o resultado da divisão do primeiro pelo segundo. Se o segundo valor informado for ZERO, deverá ser impressa uma mensagem de VALOR INVÁLIDO e ser lido um novo valor. Ao final do programa deve ser impressa a seguinte mensagem: VOCÊ DESEJA OUTRO CÁLCULO (S/N). Se a resposta for S o programa deverá retornar ao começo, caso contrário deverá encerrar a sua execução imprimindo quantos cálculos foram feitos. OBS: O programa só deverá aceitar como resposta para a pergunta as letras S ou N
2. Escrever um algoritmo que gera e escreve os 5 primeiros números perfeitos. Um número perfeito é aquele que

é igual a soma dos seus divisores. (Ex.: 6 = 1+2+3; 28= 1+2+4+7+14 etc)..

3. Em uma *lan house* existem 20 jogadores. Cada jogador tem um número de identificação e o número de vitórias no *Counter Strike*. Fazer um programa que mostre o número e o número de vitórias do jogador com o maior número de vitórias e do jogador com o menor número de vitórias.
4. Chico tem 1,5 metros e cresce 2 centímetro por ano, enquanto Zé tem 1,1 metros e cresce 3 centímetros por ano. Construa um programa que calcule e mostre quantos anos serão necessários para que Zé seja maior que Chico.
5. Tentando descobrir se um dado era viciado, um dono de cassino honesto lançou-o **n** vezes. Escreva um programa que leia um número inteiro positivo **n** e os **n** resultados dos lançamentos, e determine o número de ocorrências de cada face.
6. Fazer um programa que calcule a seguinte soma: $H = 10 + \frac{10}{2} + \frac{10}{3} + \ldots + \frac{10}{N}$. Onde o valor N é lido do teclado.
7. A Federação Paranaense de Futebol contratou você para escrever um programa para fazer uma estatística do resultado de vários ATLETIBAS. Escreva um algoritmo para ler o número de gols marcados pelo Coritiba, o número de gols marcados pelo Atlético em um ATLETIBA, imprimindo o nome do time vitorioso ou a palavra EMPATE. Logo após escrever a mensagem: "Novo ATLETIBA 1.Sim 2.Não?"e solicitar uma resposta. Se a resposta for 1, o programa deve ser executado novamente solicitando o número de gols marcados pelos times em uma nova partida, caso contrário deve ser encerrado imprimindo:
   - Quantos ATLETIBAS fizeram parte da estatística;
   - O número de vitórias do Coritiba;
   - O número de vitórias do Atlético;
   - O número de empates;
   - Uma mensagem indicando qual o time que venceu o maior número de ATLETIBAS (ou NÃO HOUVE VENCEDOR).
8. Faça um programa que leia um caractere e o número de linhas. Supondo que seja informado o caractere ”*”, seu programa deverá mostrar e 6 linhas:

```
*
**
***
****
*****
******
```

9. Idem ao anterior. Supondo que seja informado o caractere ”*”, seu programa deverá mostrar e 6 linhas:

```
**
****
******
********
**********
```

10. Idem ao anterior. Supondo que seja informado o caractere ”*”, seu programa deverá mostrar e 6 linhas:

```
******
*****
```

* * * *
* * *
* *
*