Quem pensa pouco erra muito.

Leonardo da Vinci

1 Operadores Aritméticos em *C++*

Para você calcular a vida restante de um personagem após um confronto (um golpe de espada, por exemplo) é necessário utilizar operações matemáticas. O programa 1 demonstra a utilização dos operadores matemáticos.

Programa 1: Primeiras operações matemática em C++.

```
1  // Operações matemáticas
   // Como trabalhar com os operadores
   // programa_001.cpp
   #include <iostream>
   using namespace std;
6
   int main()
   {
     cout << "7 + 3 = " << 7 + 3 << endl;
     cout << "7 - 3 = " << 7 - 3 << endl;
11   cout << "7 * 3 = " << 7 * 3 << endl;

     cout << "7 / 3 = " << 7 / 3 << endl;
     cout << "7.0 / 3.0 = " << 7.0 / 3.0 << endl;

16   cout << "7 % 3 = " << 7 % 3 << endl;

     cout << "7 + 3 * 5 = " << 7 + 3 * 5 << endl;
     cout << "(7 + 3) * 5 = " << (7 + 3) * 5 << endl;
     return 0;
21 }
```

Os operadores aritméticos da linguagem C++ estão na tabela 1:

Tabela 1: Operadores aritméticos do C++.

Operador	Significado
+	Adição
-	Subtração
*	Multiplicação
/	Divisão
%	Módulo (resto da divisão)

1.1 Divisão de inteiros e divisão de números flutuantes

Você deve ter observado os resultados diferentes nas divisões apresentadas. Observe:

```
1   cout << "7 / 3 = " << 7 / 3 << endl;
2   cout << "7.0 / 3.0 = " << 7.0 / 3.0 << endl;
```

A linguagem C++ manipula dois tipos de números. Você deve se lembrar das teorias básicas de conjunto. O C++ trabalha com os conjuntos de números inteiros e números reais. No caso, o resultado de uma operação de divisão entre dois números de um mesmo conjunto (inteiros) deve ser um número do mesmo conjunto (inteiro). Observe a figura 1:

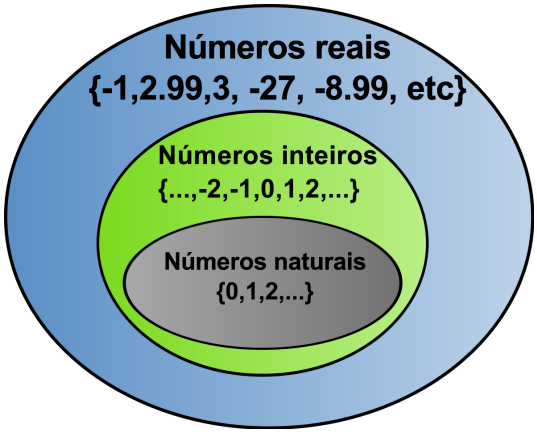


Figura 1: Teoria dos conjuntos.

Na figura 1 é possível perceber alguns conjuntos de números. O conjunto dos números naturais abragem os números começando em 0 indo e indo até o ∞ (infinito). Os números inteiros vão desde $-\infty$ até $+\infty$ e os números reais contém os conjuntos anteriores mais os números fracionários (decimais).

Por isto que uma divisão entre números inteiros sempre resultará em outro número inteiro. Mas e no caso de operações entre números inteiros e reais ? Utilize o programa 2 para chegar a uma conclusão.

Programa 2: Operações de divisão entre números reais e inteiros.

```
// Operações matemáticas
// Divisão e multiplicação entre
3   // números inteiros e reais
    // programa_002.cpp
    #include <iostream>
    using namespace std;

8   int main()
    {
      cout << "7 / 5 = " << 7 / 5 << endl;
      cout << "7 / 5.0 = " << 7 / 5.0 << endl;
      cout << "7.0 / 5 = " << 7.0 / 5 << endl;
13    cout << "7.0 / 5.0 = " << 7.0 / 5.0 << endl;

      return 0;
    }
```

1.2 Operador de resto de divisão (módulo)

Para finalizar (lembre-se do programa 1), temos o operador de resto de divisão (módulo). Representado pelo caracter %. Ele sempre retorna o resto de uma divisão por *inteiros* (logo, não se aplica a números reais).

1.3 Entendo a ordem das operações

Assim como na álgebra, a linguagem C++ avalia as expressões da esquerda para a direita. Mas alguns operadores possue maior prioridade em relação a outros. A tabela 2 mostra a prioridade de avaliação:

Tabela 2: Prioridade dos operadores em C++.

Prioridade	**Operadores**
Maior	( )
	*, /, %
Menor	+,-

Observando a tabela 2, observa-se que numa expressão $7 + 3 * 5$, primeiro ocorre a multiplicação entre os números 3 e 5, o resultado (15) é somando com 7 para obter-se o resultado final (22). Agora, numa expressão $(7 + 3) * 5$ indica que primeiro deve ocorrer a adição $(7 + 3)$ para então utilizar o resultado (10) e multiplicar por 5 (resultado obtido é 50).

2 Utilizando a memória do computador para guardar o *status* do jogo

Uma *variável* representa um pedaço da memória do computador. Utilizando variáveis, é possível guardar, recuperar ou alterar valores na memória do computador. Enfim, qualquer manipulação da memória do computador deve ocorrer utilizando-se as variáveis.

2.1 Variáveis e memória do computador

Normalmente, um programa para ser executado pelo computador possui as seguintes fases:

1. Coleta das informações de que o programa precisa;
2. Processamento;
3. Apresentação dos resultados.

Todo programa trata com informações. Um programa para computador também precisa trabalhar com informações. No computador estas informações sempre estão representadas em *variáveis*.

Para efeito de estudo de programa, a memória é o local mais importante do computador, pois é nela que ficam gravados os programas que estão em execução pelo computador e os dados que estão sendo processados por ele. A memória é o local do computador no qual os dados e instruções de um programa a ser executado são armazenados, posteriormente recuperados para processamento e para onde os resultados desses processamentos são enviados. Cada posição ou localização da memória principal é identificada por um endereço.

Quando o computador está executando um programa, todas as informações necessárias precisam estar na memória. Para o programa poder usar estas informações, ele precisa saber onde elas estão, ou seja, seu *endereço* na memória do computador.

É muito difícil (e sujeito a erros) trabalhar com estes endereços. A solução é bem simples. O que precisamos é uma forma mais natural de fazer referência àquele endereço de memória: utilizando um *nome* (observe a figura 2).

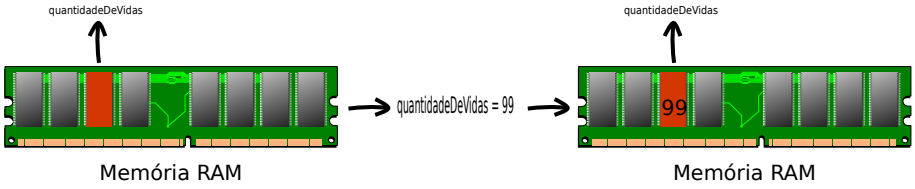


Figura 2: Processo de atribuição do valor numérico 99 à variável quantidadeDeVidas .

Logo, variáveis são espaços de memória que pode ter seu conteúdo alterado durante a execução do programa onde ela foi definida. Normalmente, são declaradas no início dos programas.

Para utilizarmos as variáveis na linguagem C++ precisamos passar pelas etapas de **declaração** e **inicialização**.

Em C++ declaramos uma variável informando o seu tipo e o seu nome (identificador):

```
1  int placar;
2  string nomeDoJogador;
3  bool luzAcesa;
```

Após declararmos uma variável, antes de utiliza-lá, devemos inicializá-la para evitarmos problemas no nosso código:

```
1  placar = 0;
2  nomeDoJogador = "Capitão Salpicão";
3  luzAcesa = false;
```

Lembre-se: Em C++ devemos informar o tipo de dado que a variável irá armazenar. Isso se deve ao fato que a linguagem é fortemente tipada, ou seja, cada variável deve estar vinculada a um determinado tipo de dado.

Pare para pensar: O que aconteceria se não inicializarmos uma variável ? Um programa em C++ permite criar um código sem inicializar uma variável antes de utilizá-la e, entre os iniciantes nessa linguagem, é um "erro" bem comum. Quando declaramos uma variável, estamos reservando um espaço numa determinada posição da memória do computador. Esse espaço poderia ter sido utilizado por outro programa anteriormente e poderia conter dados não mais utilizados. Se não inicializarmos a variável que está nessa posição acabaríamos lendo dados ou trechos de dados que não conhecemos, não sabemos o tipo, ou seja, essa informação seria uma "sujeira" que estava na memória. Se fôssemos capazes de tentar utilizar essa informação aleatória, o nosso programa poderia ter um comportamento imprevisível.

2.2 Variáveis para controlar o status de um jogo

O programa 3 utiliza variáveis para demonstrar o controle de um jogo. Neste programa é possível perceber ainda a entrada de dados pelo usuário (comando *cin*). Repare que temos o controle de vidas restantes, número de inimigos mortos, placar (*score*), se o escudo está ativo, entre outras informações. O importante é notar que: para cada informação que seja necessário manter na memória, existe uma variável associada.

2.3 Dominando os tipos básicos

Cada variável criada é tem um *tipo*, que representa o tipo da informação que você quer representar naquela variável.

A linguagem C++ define os seguintes tipos básicos de variáveis:

int : variável tipo inteira. Deve ser utilizado para se armazenar valor inteiro, com ou sem sinal. Exemplo: -1,0,1.

char : variável do tipo caracteres. Servirá para se armazenar um único caractere. Exemplo: 'a', 'A', 'f', '1'.

float : para valores com casas decimais (reais) deve-se utilizar este tipo. Ele pode armazenar números reais com até 7 dígitos significativos.

double : é o mesmo que o anterior, só que pode armazenar mais dígitos, dando uma precisão maior nos cálculos com casas decimais.

bool : variável tipo booleana (verdadeiro ou falso). Ressalta-se que na linguagem C++, em comparações lógicas, 0 (zero) é considerado falso e diferente de 0 (zero) é considerado verdadeiro

O programador pode criar novos tipos de dados, utilizando a declaração *typedef* (o que ocorre ao final do programa 3).

Programa 3: Utilização de variáveis para controlar status de um jogo.

```
// Trabalhando com variáveis
// Controle do status de um jogo
// programa_003.cpp
#include <iostream>
using namespace std;

int main()
{
  // tipo e nome das variáveis
  int score;
  double distancia;
  char jogarNovamente;
  bool escudoAtivo;

  short vidas, aliensMortos;

  score = 0;
  distancia = 1200.76;
  jogarNovamente = 's';
  escudoAtivo = true;
  vidas = 3;
  aliensMortos = 10;
```

```
  double temporario = 6572.89; // é possível criar variáveis a qualquer momento

  cout << "Score: " << score << endl;
  cout << "Distancia: " << distancia << endl;
  cout << "JogarNovamente: " << jogarNovamente << endl;

  // a impressão de um valor boleano (V ou F)
  // não é possível, no caso do valor ser V (true)
  // será impresso 1 e caso o valor seja F (false)
  // será impresso 0
  cout << "EscudoAtivo: " << escudoAtivo << endl;
  cout << "Vidas: " << vidas << endl;
  cout << "AliensMortos:" << aliensMortos << endl;
  cout << "Temporario:" << temporario << endl;

  int combustivel; // é possível declarar variáveis no meio o programa
  cout << endl; //impressão de linha em branco
  cout << "Colocar quanto combustível ?" ; // entrada de dados
  cin >> combustivel; // entrada de dados
  cout << "Combustivel:" << combustivel << endl;

  typedef unsigned short int xpto; // não se preocupe, logo você entenderá!!!
  xpto bonus = 10;

  cout << endl;
  cout << "Bonus = " << bonus << endl;

  return 0;
}
```

2.4 Abrangência dos tipos básicos

A linguagem C++ determina para cada tipo básico um certo tamanho em *bytes*. Este tamanho irá determinar a escala de valores que pode ser colocada dentro de um determinado tipo. Observe a tabela 3:

Tabela 3: Abrangência dos tipos básicos em C++.

Tipo	Abrangência
int	-2.147.483.648 até 2.147.483.647
char	256 caracteres possíveis (-127 até 128)
float	3,4E-38 até 3.4E+38 (7 dígitos de precisão)
double	1,7E-308 até 1,7E+308 (15 dígitos de precisão)
bool	*true* ou *false*

É importante ressaltar que as limitações de cada tipo está relacionada com a quantidade de bytes utilizado na memória para guarda-lo. Utilizando o operador *sizeof* é possível determinar o tamanho de cada tipo. Utilize o programa 4 para indentificar o tamanho de cada tipo básico:

Programa 4: Determinando o tamanho dos tipos básicos.

```
// Verificando o tamanho dos tipos básicos
// Uso do operador sizeof
// programa_004.cpp
#include <iostream>
using namespace std;

int main()
{
   cout << "Tamanho do char:" << sizeof(char) << endl;
   cout << "Tamanho do int:" << sizeof(int) << endl;
   cout << "Tamanho do bool:" << sizeof(bool) << endl;
   cout << "Tamanho do float:" << sizeof(float) << endl;
   cout << "Tamanho do double:" << sizeof(double) << endl;
   return 0;
}
```

2.5 Entendendo os modificadores de tipos

Você pode utilizar modificadores para alterar um tipo básico. Por exemplo, *short* é um modificador que reduz o espaço total de uma variável do tipo *int*. O modificador *long* é um modificador que aumenta o espaço total utilizando por uma variável (*int* ou *double*).

Importante: Lembre-se que vários dispositivos tem limitações de memória (celulares por exemplo). Então aprenda a utilizar pouca memória desde o início, a fim de otimizar os jogos que você desenvolverá no futuro.

Os modificadores *signed* e *unsigned* indicam se a variável irá trabalhar com números negativos (*signed* - com sinal) ou somente números positivos (*unsigned* - sem sinal). A tabela 4 demonstra a utilização de todos estes modificadores de tamanho.

Estes limites podem ser verificados no arquivo */usr/include/limits.h* do pacote C++ e são válidos para plataformas de 32 bits. Em plataformas de 16 bits, o *int* era definido com 16 bits (o equivalente a *short int* na plataforma de 32 bits). Para plataformas de 64 bits veja a tabela 5:

Tabela 4: Abrangência dos tipos básicos modificados em C++.

Tipo	Abrangência
unsigned char	256 caracteres possíveis (0 até 255)
signed char	256 caracteres possíveis (-127 até 128)
unsigned int	0 até 4.294.967.295
signed int	-2.147.483.648 até 2.147.483.647
short int	-32.768 até 32.767
signed short int	-32.768 até 32.767
unsigned short int	0 até 65.535
long int	-2.147.483.648 até 2.147.483.647
signed long int	-2.147.483.648 até 2.147.483.647
unsigned long int	0 até 4.294.967.295
long double	3,4E-4932 até 3,4E+4932 (19 dígitos de precisão)

Tabela 5: Abrangência dos tipos básicos modificados em C++ em plataformas de 64 bits.

Tipo	Abrangência
long int	-9223372036854775806 até 9223372036854775807
signed long int	-9223372036854775806 até 9223372036854775807
unsigned long int	0 até 18446744073709551615

O termo *plataforma*, utilizado neste texto, refere-se ao suporte do processador em conjunto com o sistema operacional. Por exemplo: um sistema operacional de 64 bits rodando sob um processador com suporte a 64 bits.

Para determinar o tamanho em bytes de cada tipo modificado, compare o resultado do programa 4 com o programa 5.

Programa 5: Determinando o tamanho dos tipos básicos modificados.

```
// Verificando o tamanho dos tipos básicos
// Uso do operador sizeof
// programa_005.cpp
#include <iostream>
```

```
5  using namespace std;

   int main()
   {

10   cout << "Tamanho do unsigned char:" << sizeof(unsigned char) << endl;
     cout << "Tamanho do signed char:" << sizeof(signed char) << endl;
     cout << "Tamanho do unsigned int:" << sizeof(unsigned int) << endl;
     cout << "Tamanho do signed int:" << sizeof(signed int) << endl;
     cout << "Tamanho do short int:" << sizeof(short int) << endl;
15   cout << "Tamanho do signed short int:" << sizeof(signed short int) << endl;
     cout << "Tamanho do unsigned short int:" << sizeof(unsigned short int) << endl;
     cout << "Tamanho do long int:" << sizeof(long int) << endl;
     cout << "Tamanho do signed long int:" << sizeof(signed long int) << endl;
     cout << "Tamanho do unsigned long int:" << sizeof(unsigned long int) << endl;
20   cout << "Tamanho do long double:" << sizeof(long double) << endl;
     return 0;
   }
```

No C++, você pode abreviar a utilização dos tipos. Por exemplo, utilizar *short* tem o mesmo significado que *short int*. Assim como utilizar *long* tem o mesmo significado que *long int*.

2.6 Declarando as variáveis

Para se usar uma variável em C++, ela deve ser definida indicando o seu tipo e o seu nome. Por exemplo, a declaração:

```
1  int score;
```

Indica que estamos criando a variável *score* e que ela é do tipo *int*. Devemos utilizar o nome da variável para acessa-la.

No programa 3 declaramos vários tipos de variáveis. Jogos e outras aplicações utilizarão muitas variáveis, mas felizmente a linguagem C++ permite que sejam declaradas várias variáveis do mesmo tipo ao mesmo tempo. Por exemplo:

```
1  short vidas, aliensMortos;
```

2.7 O poder dos nomes de variável

Escolher corretamente o nome das variáveis e rotinas (principalmente variáveis) é mais um assunto em programação que parece não fazer tanta diferença, ainda mais para os iniciantes, onde os programas são pequenos e feitos rapidamente, porém, em se tratando de programas maiores, que são muito mais demorados para se fazer, chamar uma variável de *zelda*, *a*, *b* ou *c* pode ocasionar vários problemas.

Toda variável de um programa deve ter um nome único dentro do contexto de existência dela. Para se formar o nome de uma variável deve-se seguir algumas regras impostas pelo compilador. Estas regras são:

1. O nome de uma variável deve começar por uma letra ou por um caractere "_"(*underline*);
2. Os demais caracteres de uma variável podem ser letras, dígitos e "_";
3. O compilador reconhece os primeiros 31 caracteres para diferenciar uma variável de outra;
4. Um ponto importante a ser ressaltado é que para o compilador C++ as letras maiúsculas são diferentes das letras minúsculas;
5. No C++, o nome de cada variável começa com uma letra em minúscula. No caso de nome composto, as próximas letras de cada nome deverão ser em maiúsculas.

```
1 float soma; //válido e dentro do padrão
2 int quantidadeDeVidas; //válido e dentro do padrão
```

O processo de escolha de nome de uma variável é importante para a legibilidade de um jogo em manutenções posteriores. Algumas regras básicas que se seguidas irão melhorar muito a futura manutenção do seu jogo.

- Não utilize nomes que não tenham significados com o uso da variável. Por exemplo: uma variável com o nome *Cont* utilizada para se guardar a soma de um procedimento. Melhor seria utilizar uma variável com o nome de *Soma*.
- Se uma variável for utilizada para guardar a soma de um valor, como por exemplo, o total de pontos obtidos ao se abater inimigos, além da função coloque também o conteúdo da mesma, chamando a variável de *SomaTotalPontos*.

A linguagem C++ utiliza o padrão *CamelCase*, ou seja, todo nome de variável começa com uma letra em minúscula e no caso de um nome composto, as próximas letras de cada nome serão em maiúsculas.

```
1 int totalVidas; //válido e dentro do padrão
2 int TotalVidas; // válido mas fora do padrão
3 int NúmeroDeAcertos // inválido, contém acentos
4 char 2Cor; //inválido, começa com um número
5 float float; //inválido, utiliza uma palavra reserva da linguagem
6 double Quantidade de Gas; //inválido, contém espaços em branco
7 long umaVariavelComUmNomeSuperHiperMegaUltraGigante; //válido
8 long #telefone; //inválido pois o caracter # não pode ser utilizado
```

Apenas reforçando, a linguagem C++ diferencia letras maiúsculas (caixa alta) de minúsculas (caixa baixa), logo, na declaração a seguir temos três variáveis distintas na memória do computador:

```
1  int TotalVidas;
2  int TOTALVidas;
3  int totalVidas;
```

2.8 Atribuição de Valores

Ás vezes ao se definir uma variável é desejável que a mesma já tenha um valor pré-definido. A linguagem C++ permite que quando se defina uma variável se indique também o valor inicial da mesma. Deve-se colocar após a definição da variável o caractere "="seguido de um valor compatível com o tipo da variável.

Assinalando várias do tipo *int* : como visto anteriormente (tabelas 3 e 4), uma variável inteira recebe valores do conjunto de números inteiros (lembre-se da figura 1).

```
1  score = 0;
```

Assinalando várias do tipo *float* ou *double* : estes tipos de dados recebem valores reais, neste caso, deve-se sempre informar o número com suas respectivas casas decimais. Na linguagem C++, utiliza-se o . (ponto) como separador de casas decimais.

```
1  distancia = 1200.76;
```

Assinalando variáveis do tipo *char* : o valor a ser assinalado para o tipo *char* deve estar entre aspas simples (”).

```
1      jogarNovamente = 's';
```

Variáveis do tipo *char* podem armazenar 128 valores da tabela *ASCII* (*American Standard Code for Information Interchange*). Entre estes valores está a faixa entre ’a’ até ’z’, ’A’ até ’Z’ e ’0’ até ’9’.

Assinalando variáveis do tipo *bool* (booleanas) : variáveis deste tipo recebem apenas dois tipos de valores: *true* ou *false*.

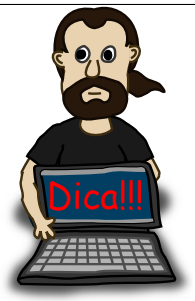
```
1      escudoAtivo = true;
```

Variáveis deste tipo são muito importantes em processos de decisão e repetição (que serão vistos nas próximas aulas).

A iniciação e declaração da variável devem serem feitas, preferencialmente, perto de onde ela for usada pela primeira vez. Isso poupará problemas quando for preciso, em códigos mais extensos, localizar a variável e alterar o seu valor, porém certifique-se de que os valores atribuídos são razoáveis e fazem sentido no programa, por exemplo, colocar valor inicial 0 para uma variável multiplicadora fará com que todos os elementos multiplicados sejam igual a zero. Deve-se tomar cuidado, também, com impressão estética, pois declarar todas as variáveis logo no inicio do programa faz parecer que todas serão utilizadas na função logo abaixo ou serão utilizadas em todo o programa. Observe o trecho:

```
1  int main()
2  {
```

```
3     int numeroDeInimigos=30;
4     //código utilizando a variável numeroDeInimigos
5     ...
6     ...
7     int qtdMunicao=5000;
8     //código utilizando a variável qtdMunicao
9     ...
10    ...
11 }
```



Deve-se sempre lembrar que o nome de uma variável e a sua função devem representar a mesma coisa, uma variável com nome ruim pode ser determinante para que o desempenho de uma variável seja ruim. Como no trecho de código:

```
1  int main()
2  {
3     ...
4     a=b+c;
5     d=a/2;
6     ...
7  }
```

Repare como o trecho de código anterior está *confuso*, apesar de parecem estarem relacionadas, não é possível saber qual é o significado de cada variável e muito menos a finalidade do código. Variáveis com nomes simples são fáceis de memorizar e deixam o código muito mais legível. Observe outra versão do trecho de código, agora com nomes simples e fácies e que expõem o propósito do código:

```
1  int main()
2  {
3     ...
4     somaPontos=pontosA+pontosB;
5     media=somaPontos/2;
6     ...
7  }
```

2.9 Declaração *typedef* ou como criar seus próprios tipos de dados

Na linguagem C++ pode-se dar um outro nome a um tipo determinado. Isto é feito através da declaração *typedef*. É muito usado para se manter a compatibilidade entre os sistemas operacionais e também para encurtar algumas definições longas, simplificando o programa.

```
1  typedef unsigned short int ushort;
2  ushort bonus = 10; // e como simplifica nao precisar digitar tudo.
3  ushort vidasExtras = 1;
```

O *typedef* é muito usado para se criar tipos na própria língua do programador, criando-se tipos equivalentes.

3 Entrada de dados pela entrada padrão

O comando (objeto) *cin* deve ser usado para se realizar a entrada de dados da entrada padrão do sistema operacional para as variável indicadas.

A análise do *buffer* de entrada será realizada de acordo com o tipo de variável indicada no comando (isto ocorre de forma automática). Ou seja, se a variável de entrada for do tipo inteiro, o comando *cin* interpretará o que está sendo digitado como inteiro.

A indicação dos valores que devem ser impressos via comando *cin* precisam utilizar o operador de entrada >>. Cada variável deve ser precedido por este operador. Pode-se colocar mais de um operador de entrada para um único *cin*. As informações serão lidas em seqüência. A separação dos argumentos de entrada, normalmente, é feita através do separador branco. Veja exemplos de uso do comando *cin* no programa 6.

Programa 6: Trabalhando com a entrada padrão.

```
// Trabalhando com variáveis
2  // Entrada de dados
   // programa_006.cpp
   #include <iostream>
   using namespace std;

7  int main()
   {
     // tipo e nome das variáveis
     int score;
     double distancia;
12   char jogarNovamente;
     bool escudoAtivo;

     short vidas;

17   cout << "Entre com o score:";
     cin >> score;
     cout << "Entre com a distância e prentende jogar novamente:";
     cin >> distancia >> jogarNovamente; //lê duas informações ao mesmo tempo
     cout << "Informe se o escudo está ativo:";
22   cin >> escudoAtivo; //0 indica falso e qualquer outro valor indica verdadeiro
     cout << "Informe a quantidade de vidas:";
     cin >> vidas;
```

```
  cout << endl; //linha em branco
  cout << "Score: " << score << endl;
  cout << "Distancia: " << distancia << endl;
  cout << "JogarNovamente: " << jogarNovamente << endl;

  // a impressão de um valor boleano (V ou F)
  // não é possível, no caso do valor ser V (true)
  // será impresso 1 e caso o valor seja F (false)
  // será impresso 0
  cout << "EscudoAtivo: " << escudoAtivo << endl;
  cout << "Vidas: " << vidas << endl;

  return 0;
}
```

Lembre-se de sempre ter um valor atribuido a uma variável. Você pode atribuir um valor diretamente ou ler da entrada padrão. Tome cuidado para que não ocorra situações como esta:

```
  int totalVidas;
  cout << "Total de Vidas:" << totalVidas << endl;
```

O código acima funciona, mas o resulta é imprevisível, pois *totalVidas* não recebeu nenhum valor, ou seja, não foi atribuido um valor com = para à variável e muitos menos houve uma entrada de dados pelo teclado.

4 Operações aritméticas utilizando variáveis

Observe o programa 7. Neste programa pode-se observar várias operações ariméticas. Além dos operadores já visto na tabela 1, temos outros operadores: *operadores unários* e *operadores de atribuição*.

Programa 7: Trabalhando com a entrada padrão.

```
// Variáveis e operações matemáticas
// Outro controle do status de um jogo
// programa_007.cpp
#include <iostream>
using namespace std;

int main()
{
  unsigned int score = 5000;
  cout << "Score:" << score << endl;

  //alterando o valor da variável
  score = score + 10;
  cout << "Score:" << score << endl;

  //combinando o operador de soma
```

```
  score += 100; //equivale a score = score + 100;
  cout << "Score:" << score << endl;

  //combinando o operador de subtração
  score -= 20; //equivale a score = score - 20;
  cout << "Score:" << score << endl;

  // utilizando o operador de incremento
  int vidas = 3;
  ++vidas;
  cout << "Vidas:" << vidas << endl;
  vidas++;
  cout << "Vidas:" << vidas << endl;

  vidas = 3;
  int bonus;
  bonus = ++vidas * 10;
  cout << "Vidas, Bonus = " << vidas << ", " << bonus << endl;

  vidas = 3;
  bonus = vidas++ * 10;
  cout << "Vidas, Bonus = " << vidas << ", " << bonus << endl;

  //ultrapassando o valor máximo
  score = 4294967295;
  cout << "Score:" << score << endl;
  ++score;
  cout << "Score:" << score << endl;
  --score;
  cout << "Score:" << score << endl;
  return 0;
}
```

4.1 Operadores de atribuição

Como é comum a atribuição aonde uma variável é somada ou diminuída de um valor e o resultado é atribuído à mesma variável, a Linguagem C++ disponibiliza uma maneira curta de realizar este tipo de operação (veja a tabela 6).

Tabela 6: Operadores de atribuição.

Operador	Operação	Exemplo	Equivalência
$+=$	Adição	vidas $+=1$;	vidas = vidas $+$ 1;
$-=$	Subtração	vidas $-=1$;	vidas = vidas $-$ 1;
$*=$	Multiplicação	bonus $*=10$;	bonus = bonus $*$ 10;
$/=$	Divisão	danoRecebido $/=3$;	danoRecebido $=$ danoRecebido$/3$;
$\%=$	Módulo	custoSkill $\%=3$;	custoSkill $=$ custoSkill $\%$ 3;

4.2 Operadores Unários

A posição relativa destes operadores (tabela 7) em relação a variável influencia o seu funcionamento. Se os operadores forem colocados antes da variável em uma expressão, inicialmente será efetuado o incremento e depois será utilizado este novo valor na expressão. Se os operadores forem colocados após a variável, primeiro será utilizado o valor atual da variável na expressão, e após, será realizado o incremento ou decremento.

Tabela 7: Operadores de unário.

Operador	Operação	Exemplo	Equivalência
$++$	Incremento	proximoLevel++;	proximoLevel = proximoLevel + 1;
$--$	Decremento	vidas−−;	vidas = vidas − 1;

Na sequência abaixo (retirada do programa 7), o *operador unário* de incremento está na frente da variável *Vidas*. Isto indica ao programa que: 1°soma-se 1 em *Vidas* (resultando em 4) e 2°multiplica-se o valor obtido por 10 (resultado em 40).

```
1  vidas = 3;
2  int bonus;
3  bonus = ++vidas * 10;
4  cout << "Vidas, Bonus = " << vidas << ", " << bonus << endl;
```

Na prática, o trecho de código acima seria o equivalente a:

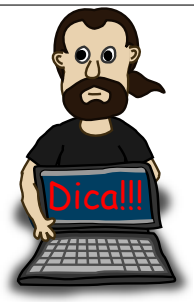
```
1  vidas = 3;
2  int bonus;
3  vidas = vidas + 1; // primeiro soma
4  bonus = vidas * 10; // depois multiplica
5  cout << "Vidas, Bonus = " << vidas << ", " << bonus << endl;
```

Por analogia, na próxima sequência, como o *operador unário* de incremento está depois da variável *vidas*: 1°multiplica-se o valor de *Vidas* por 10 (resultando em 30) e 2°soma-se 1 em *vidas* (resultando em 4).

```
1  vidas = 3;
2  bonus = vidas++ * 10;
3  cout << "Vidas, Bonus = " << vidas << ", " << bonus << endl;
```

Na prática, o trecho de código acima seria o equivalente a:

```
1  vidas = 3;
2  bonus = vidas * 10; // primeiro multiplica
3  vidas = vidas + 1; // depois soma
4  cout << "Vidas, Bonus = " << vidas << ", " << bonus << endl;
```



Dica de Performance: Existe diferença entre a utilização *vidas*++ (pós-incremento) para ++*vidas* (pré-incremento). Ambos incrementam *vidas* em 1, mas no pós-incremento:

1. Ocorre uma cópia de *vidas*;
2. *vidas* é incrementado em 1;
3. É retornado o valor da cópia de *vidas*.

No pré-incremento ocorre apenas o incremento de *vidas*.

Lembre-se dos valores neutros para multiplicação, divisão, soma e subtração. Ao criar variáveis, lembre-se sempre de começar com um valor neutro:

```
1  int soma = 0; //0 é elemento neutro da soma
2  int diminui = 0; //0 é o elemento neutro da subtração
3  int multiplica = 1; //1 é o elemento neutro da multiplicação
4  int divide = 1; //1 é o elemento neutro da divisão
```

4.3 Ultrapassando o limite de uma variável

Durante uma operação de incremento, se o valor máximo, que uma variável pode conter, é ultrapassado, não ocorre erro algum. A variável simplesmente recebe o valor *minímo*. De forma análoga ocorre se uma variável estiver com o valor minímo e ocorrer uma operação de decremento. Neste caso a variável recebe o valor *máximo*.

```
1  score = 4294967295;
2  cout << "Score:" << score << endl;
3  ++score;
4  cout << "Score:" << score << endl;
5  --score;
6  cout << "Score:" << Score << endl;
```

4.4 Linearização de expressões

Para a construção de programas todas as expressões aritméticas devem ser linearizadas, ou seja, colocadas em linhas. É importante também ressalvar o uso dos operadores correspondentes da aritmética tradicional para a computacional. Além disso devemos tomar cuidado com a ordem de precedência matemática de cada operador.

Considere a expressão: $\frac{2}{3} + (5 - \frac{3}{7})$.

Um computador não entende esta expressão, pois não está linearizada. A mesma expressão linearizada ficaria assim: $(2/3) + (5 - (3/7))$. Desta forma, o computador entede a expressão, pois estamos utilizando os símbolos corretos, ao mesmo tempo incluímos alguns parênteses para melhor entendimento do programador.

Numa equação ou representação simbólica, existe uma ordem de avaliação ou processamento. O computador passa analisando e resolvendo a representação simbólica obedecendo a ordem de precedência.

5 Entendendo o tamanho de uma variável

Um *byte* é um dos tipos de dados integrais em computação. É usado com frequência para especificar o tamanho ou quantidade da memória ou da capacidade de armazenamento de um computador, independentemente do tipo de dados armazenados. Cada *byte* é representando por 8 bits (menor unidade de informação que pode ser armazenada ou transmitida). Um bit pode assumir somente 2 valores: 0 ou 1.

Para os computadores, representar 256 números, um byte é suficiente. Como um bit representa dois valores (1 ou 0) e um byte representa 8 bits, basta fazer 2 (do bit) elevado a 8 (do byte) que é igual a 256. Veja a figura 3:

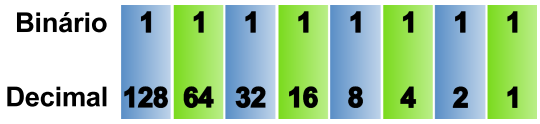


Figura 3: Representação binária do número 255.

Um bit 1 representa que ele está ligado naquela posição. Um bit zero representa que está desligado naquela posição. Como $2^8 = 256$ (base binário e 8 bits) a soma de todos os bits ativos indica o valor representado pelo byte. Observe a tabela 8:

Tabela 8: Valores binários.

Posição	Cálculo	Valor decimal
8º	2^7	128
7º	2^6	64
6º	2^5	32
5º	2^4	16
4º	2^3	8
3º	2^2	4
2º	2^1	2
1º	2^0	1

Neste momento você está pensando: "*Se* $2^8 = 256$, *então qual o motivo de todos os bits ligados (1) representarem o valor decimal 255 ?*". Simples, todos os bits desligados (0) representa o valor zero. Por isto que um byte representa 256 números possíveis. A figura 4 demonstra a representação binária do valor decimal 65.

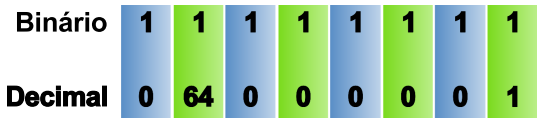


Figura 4: Representação binária do número 65.

5.1 Conversão da base decimal para a base binária

A conversão do número inteiro, de decimal para binário, será feita da direita para a esquerda, isto é, determina-se primeiro o algarismos das unidades (o que vai ser multiplicado por 2^0) , em seguida o segundo algarismo da direita (o que vai ser multiplicado por 2^1) etc...

A questão chave, por incrível que pareça, é observar se o número é par ou ímpar. Em binário, o número par termina em 0 e o ímpar em 1. Assim determina-se o algarismo da direita, pela simples divisão do número por dois; se o resto for 0 (número par) o algarismo da direita é 0; se o resto for 1 (número ímpar) o algarismo da direita é 1.

Por outro lado, é bom lembrar que, na base dez, ao se dividir um número por dez, basta levar a vírgula para a esquerda. Na base dois, ao se dividir um número por dois, basta levar a vírgula para a esquerda. Assim, para se determinar o segundo algarismo, do número em binário, basta lembrar que ele é a parte inteira do número original dividido por dois, abandonado o resto.

Vamos converter 25 e 132 de decimal para binário (figura 5). Repare que os números em vermelho são os restos das diversas divisões ocorridas. O número binário é obtido lendo os restos de baixo para cima.

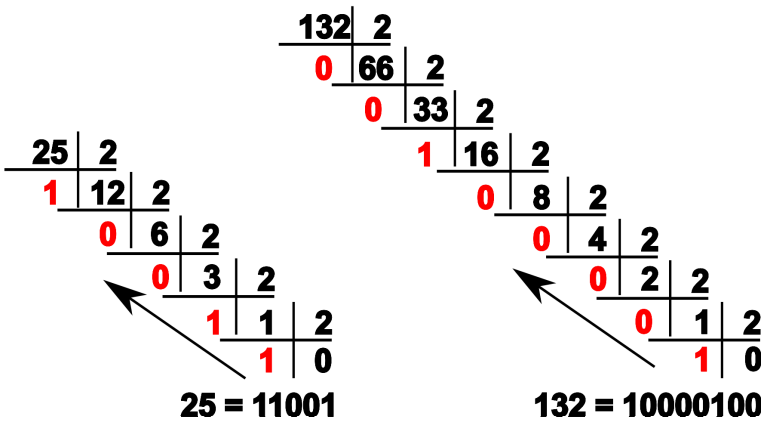


Figura 5: Conversão dos números 25 e 132 de decimal para binário.

6 Conversão de Tipos

De uma forma geral pode-se realizar a conversão de um tipo para outro da linguagem C++ utilizando o que se chama de *typecast*. Esta técnica é muito utilizada para se melhorar o entendimento de alguns trechos de programas. Outras vezes utiliza-se o *typecast* para compatibilizar um determinado tipo de um parâmetro na chamada de uma função para o tipo do parâmetro esperado por aquela função.

A sintaxe do *typecasting* é:

```
(tipo) valor_constante;
(tipo) variável ;
```

- No primeiro formato o compilador irá realizar a transformação da constante indicada para o tipo indicado entre parênteses durante o processo de compilação.
- No segundo formato o compilador irá gerar o código adequado para que a conversão ocorra em tempo de execução.

O programa 8 apresenta alguns exemplos de *casting*.

Programa 8: Exemplos de *casting* entre variáveis.

```
// mostrar a conversão entre tipos de variáveis
// programa_008.cpp
#include <iostream>
using namespace std;

int main()
{
  char caracter = 'E';
  int ascii = (int) caracter; // Código ASCII do 'E'

  cout << "Conversão de caracter para inteiro:";
  cout << caracter << " = " << ascii << endl;

  cout << "Conversão de inteiro para caracter:";
  ascii = 65;
  caracter = (char) ascii;
  cout << ascii << " = " << caracter << endl;


  short int valor = 10;
  float resultado = 7.0;
  int somaInteiro;
  float somaFloat;
  somaInteiro = (int) (valor/resultado); // converter o resultado de uma divisao para inteiro

  somaFloat = valor/resultado; //lembrar que a divisão de um inteiro por
                        //um float sempre retorna um float
  cout << "Valor convertido para int:" << somaInteiro << endl;
  cout << "Valor que seria retornado (float):" << somaFloat << endl;

  somaFloat = valor/(int)resultado; //convertendo o float para int antes da divisão
  cout << "Valor retornado:" << somaFloat << endl;
  cout << "Game over!!" << endl;
  return 0;
}
```

7 Para facilitar nossa vida - uma biblioteca de funções

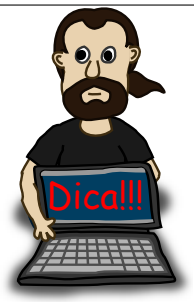
A fim de agilizar o aprendizado da lógica e não perdermos tempo com possíveis problemas de compatibilidade de sistemas operacionais e diferentes compiladores C++, um conjunto de funções foi criado e disponibilizado para uso. Todas as funções estão descritas no Anexo **??** e serão apresentadas a medida que formos progredindo no desenvolvimento da lógica de programação. O primeiro conjunto de funções diz respeito a leitura de dados do teclado, trabalhar com cores, conversão de números para texto e ler o conteúdo de um arquivo:

- *mudaCor(NumeroCorLetra)* - muda a cor da fonte de uma mensagem;
- *mudaCor(NumeroCorLetra, NumeroCorFundo)* - muda a cor da fonte e do fundo de uma mensagem;
- *limpaEfeito()* - limpa efeito de cores;
- *limparTela()* - limpa a tela;
- *readInt()* - lê um número inteiro do teclado;
- *readInt(Mensagem)* - mostra uma mensagem e lê um número inteiro do teclado;
- *readFloat()* - lê um número decimal do teclado;
- *readFloat(Mensagem)* - mostra uma mensagem e lê um número decimal do teclado;
- *readDouble()* - lê um número decimal do teclado;
- *readDouble(Mensagem)* - mostra uma mensagem e lê um número decimal do teclado;
- *readString()* - lê uma frase do teclado;
- *readString(Mensagem)* - mostra uma mensagem e lê uma frase do teclado;
- *readBool()* - lê um valor lógico do teclado;
- *readBool(Mensagem)* - mostra uma mensagem e lê um valor lógico do teclado;
- *readChar(Mensagem)* - mostra uma mensagem e lê um valor caractere do teclado;
- *readChar()* - lê um valor caractere do teclado;
- *string numeroToString(valor)* - converte o valor passado para string (int, double ou float);
- *void espera(long int tempo)* - informa ao programa para *dormir* por um período de tempo, o tempo é dado em centésimos de segundo.
- *string retornaConteudoArquivo(string nomeArquivo)* - retorna o conteúdo de um arquivo texto.

7.1 Utilizando a biblioteca

Para utilizar a biblioteca, é necessário colocar diretiva no início do seu programa.

```
1 #include "biblaureano.h"
```



Utilizando a biblioteca e incluindo a diretiva no início do programa:

```
1 #include "biblaureano.h"
```

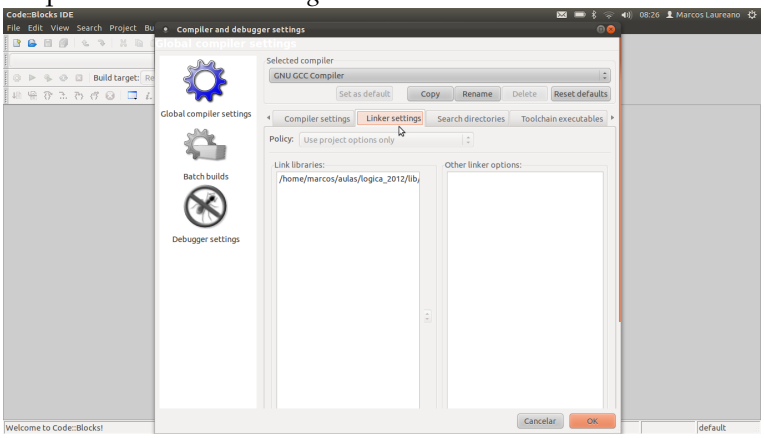
Você não precisará mais colocar os comandos:

```
1 #include <iostream>
2
3 using namespace std;
```

A biblioteca além de esconder alguns requisitos técnicos da linguagem, para acelerar o aprendizado da lógica, também evita que você precise ficar digitando trechos de programa desnecessários.

E ter o arquivo liblaureano.so (Linux) ou liblaureano.dll (Windows) configurado no CodeBlocks como biblioteca:

1. Clique na opção *Settings* e depois em *Compiler and debugger...*
2. Clique na aba *Linker settings*



3. Adicione o arquivo liblaureano.so
4. Compile o seu programa de teste.

7.2 Lendo valores do teclado

Estas funções de leitura visam evitar problemas com buffer de teclado, principalmente para a leitura de strings.

Programa 9: Impressão de Cores

```
//leitura de dados no teclado
//programa_009.cpp
#include "biblaureano.h"

int main()
{
  int x = readInt("Teste de inteiro:");
  cout << x << endl;
  x = readInt("Teste de inteiro:");
  cout << x << endl;
  string s = readString("Teste de string:");
  cout << "[" << s << "]"<< endl;
  float f = readFloat("Teste de float:");
  cout << f << endl;
  s = readString("Teste de string:");
  cout << "[" << s << "]"<< endl;
  s = readString("Teste de string:");
  cout << "[" << s << "]"<< endl;
  cout << "teste double:";
  double d = readDouble();
  cout << d << endl;
  bool b = readBool("Valor logico:");
  cout << b << endl;
  return 0;
}
```

7.3 Convertendo números para string

Esta função visa facilita a conversão de valores do tipo int, float ou double para string.

Programa 10: Conversão de números para string

```
//conversão de números
//programa_010.cpp
#include "biblaureano.h"

int main(void)
{
  int numeroInt = readInt("Entre com um valor int:");
  string convertido = numeroToString(numeroInt);
  cout << "Valor convertido para string:" << convertido << endl;

  float numeroFloat = readFloat("Entre com um valor float:");
  convertido = numeroToString(numeroFloat);
  cout << "Valor convertido para string:" << convertido << endl;

  double numeroDouble = readDouble("Entre com um valor double:");
  convertido = numeroToString(numeroDouble);
  cout << "Valor convertido para string:" << convertido << endl;

  cout << "Game over!!" << endl;
```

```
20    return 0;
}
```

7.4 Mudança de Cor

As cores permitidas para uso são dadas pela definição:

```
enum COR {BLACK, RED, GREEN, YELLOW, BLUE, PURPLE, CYAN, WHITE};
```

A mudança de cor deverá ocorrer antes da mensagem ser impressa, veja o exemplo:

Programa 11: Impressão de Cores

```
//impressão de cores
//programa_011.cpp
#include "biblaureano.h"

int main()
{
   mudaCor(GREEN);
   cout << "Verde!";
   mudaCor(YELLOW);
   cout << "Amarelo!";
   mudaCor(BLUE);
   cout << "Azul!";
   mudaCor(CYAN);
   cout << "Ciano!";
   mudaCor(PURPLE);
   cout << "Roxo!";
   mudaCor(WHITE);
   cout << "Branco!";

   mudaCor(YELLOW,BLUE);
   cout << endl << "Amarelo com Azul" << endl;
   limpaEfeito();
   cout << endl << "Mensagem sem efeitos" << endl;
   mudaCor(BLUE, YELLOW);
   cout << endl << "Azul com Amarelo" << endl;
   mudaCor(BLACK, RED);
   cout << endl << "Preto com Vermelho" << endl;
   limpaEfeito();
   cout << endl << "Mensagem sem efeitos" << endl;

   cout << "Fim do programa de exemplo." << endl;
   return 0;
}
```

7.5 Lendo o conteúdo de um arquivo

Vamos supor que você resolva realizar uma animação na tela. Por exemplo, um avião e nuvens deslocando-se pelo céu. O programa 12 lê o conteúdo de 5 arquivos em sequência para mostrar uma animação na tela.

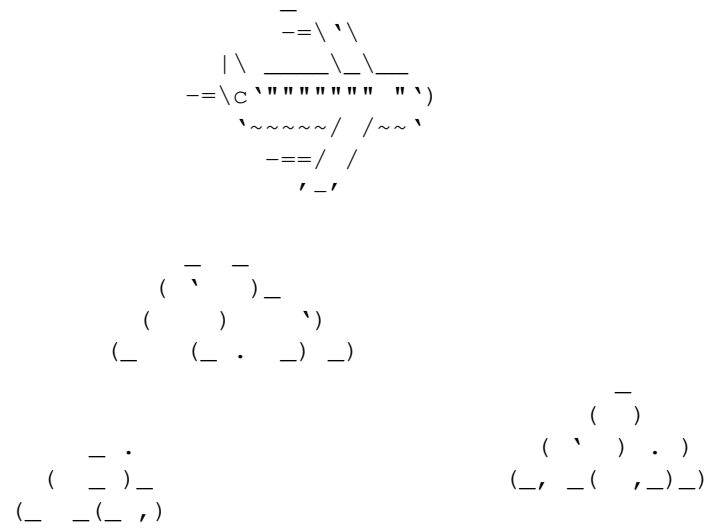
Programa 12: Lendo arquivos para simular uma animação.

```
//mostrar o conteúdo de arquivos na tela.
//programa_012.cpp
#include "biblaureano.h"

int main()
{

   cout << retornaConteudoArquivo("aviaonuvem.txt");
   espera(100); //pausa 1 segundo
   limparTela();
   cout << retornaConteudoArquivo("aviaonuvem1.txt");
   espera(100); //pausa 1 segundo
   limparTela();
   cout << retornaConteudoArquivo("aviaonuvem2.txt");
   espera(100); //pausa 1 segundo
   limparTela();
   cout << retornaConteudoArquivo("aviaonuvem3.txt");
   espera(100); //pausa 1 segundo
   limparTela();
   cout << retornaConteudoArquivo("aviaonuvem4.txt");
   espera(100); //pausa 1 segundo
   return 0;
}
```

O resultado do programa 9 é semelhante ao desenho abaixo:



8 Exercícios

O objetivo geral desta sequência de exercícios é praticar o uso da biblioteca para leitura dos dados a partir do teclado, perceber as diferenças entre números inteiros (int) e reais (float) e trabalhar com os diversos operadores matemáticos apresentados. Fica a seu critério utilizar as funções de cores para *embelezar* o seu programa.

1. Elabore um programa que leia 2 valores A e B e faça a permuta entre eles, ou seja, A recebe o valor de B e vice-versa.
2. Um professor atribui pesos de 1 a 4 para as notas de quatro avaliações. A nota é calculada por meio da média ponderada (N1 + N2*2 + N3*3 + N4*4)/10, onde N1 é a nota da primeira avaliação, N2 a da segunda, etc. Um aluno tirou as seguintes notas: 8 - 7,5 - 10 - 9. Faça um programa que calcula e mostra as notas e a média deste aluno.
3. Vou e volto diariamente a pé para o trabalho, que dista aproximadamente 800 m de minha casa. Supondo que trabalho 5 dias por semana, 45 semanas por ano, *bole* a operação matemática que deve ser efetuada para calcular quantos quilômetros, aproximadamente, terei andado ao final de um ano. Elabore um programa que faça as contas e mostre o resultado na tela.
4. Elabore um programa que calcule a quantidade de pontos quem jogador recebe ao matar diversos tipos de *monstros*. Considere que exista 3 tipos de monstros (feio, o muito feio e o horrível). O primeiro vale 1 ponto, o segundo 5 pontos e o terceiro 10 pontos. Leia a quantidade de monstros mortos de cada tipo e calcule os pontos obtidos. Ao final, calcule um bônus de 10% sobre os pontos.
5. Faça um programa que leia 5 números e mostre sua soma na tela.
6. Faça um programa que leia o raio de uma circunferência e calcule a sua área. Utilize a fórmula πR^2.
7. Escrever programa que lê três notas inteiras e calcula a sua média aritmética.
8. Elabore um programa que lê três valores e calcula a média geométrica dos números lidos (divisão do produto pela quantidade de valores).
9. Implemente um programa que lê três valores e calcule a média ponderada para pesos 1, 2 e 3, respectivamente (multiplique cada nota pelo seu peso, some os produtos e divida o resultado pela soma dos pesos).
10. Implemente um programa que lê dois números quaisquer e informa sua soma, diferença, produto e divisão.
11. O critério de notas de uma faculdade consiste de uma nota de 0 a 10 em cada bimestre, sendo a primeira nota peso 2 e a segunda peso 3. Elabore um programa que lê as notas bimestrais e calcula a nota do semestre.
12. Um canal de notícias internacionais, a cabo, previa temperatura máxima para Brasília de 85 graus Fahreneit. Escrever um programa que lhe permita converter esta temperatura (e qualquer outra) para graus Celsius, sabendo que a relação entre elas é $^\circ\mathrm{C} = \frac{5}{9} * (^\circ\mathrm{F} - 32)$.
13. Um casal divide as despesas domésticas mensalmente. Durante o mês cada um anota seus gastos e as contas que paga; no final eles dividem meio a meio. O casal deseja um programa que facilite o acerto: eles digitariam os gastos de cada um, e o programa mostraria quem deve a quem. Atualmente eles fazem o acerto manualmente, na forma da seguinte tabela:

    Portanto, os saldos devem ser iguais, e quem tiver o saldo negativo deve pagar o valor para o outro. Faça um programa que leia os valores adequados e efetue os cálculos. O total é a soma das despesas individuais; um percentual é o gasto individual dividido pelo total, multiplicado por 100; o valor devido por cada um é o mesmo e igual à metade do total; finalmente, cada saldo corresponde a diferença entre o valor pago pela pessoa e a média do que foi pago. Uma tela para o programa pode ser, com os mesmos dados da tabela acima:

ITEM	MARIDO	ESPOSA	TOTAL
Despesas Pagas	1278,60	875,30	2.153,90
% Pago	59,36	40,64	100
Valor Devido	1.076,95	1.076,95	2.153,90
Saldo	201,65	-201,65	

```
Digite valor das despesas do marido: 1278.60
Digite valor das despesas da esposa: 875.30
ITEM              MARIDO   ESPOSA     TOTAL
===============   =======  =======    =======
Despesas pagas    1278.60   875.30    2153.90
% pago              59.36    40.64        100
Valor devido      1076.95  1076.95    2153.90
Saldo              201.65  -201.65
```

14. Altere o programa acima de forma que o marido arque com 60% das despesas e a esposa com o restante.
15. Para o mesmo programa de rateio acima, suponha que o casal, ao invés de dividir meio a meio as despesas, vai dividi-las proporcionalmente à renda de cada um. Altere o programa de forma que este leia também a renda de cada um e use a proporção das rendas para a divisão.
16. Elabore um programa que dado um número inteiro de segundos, mostrar a quantas horas, minutos e segundos ele corresponde.
17. O governo acaba de liberar 10 milhões de dólares para a construção de casas populares, a qual contratou a Construtora Pica-Pau S/A. Cada casa custa o equivalente a 150 salários mínimos. Faça um algoritmo que leia o valor do salário mínimo, o valor do dólar e calcule a quantidade de casas possíveis de se construir.
18. Escrever um programa que leia um valor em reais e calcule qual o menor número possível de notas de 100, 50, 20, 10, 5 e 1 em que o valor lido pode ser decomposto e escreva o valor lido e a relação de notas necessárias.
19. Escrever um programa que leia:
    - a percentagem do IPI a ser acrescida no valor das peças
    - o código da peça 1, valor unitário da peça 1, quantidade de peças 1
    - o código da peça 2, valor unitário da peça 2, quantidade de peças 2

    O programa deve calcular o valor total a ser pago e apresentar o resultado. Fórmula: Total = (valor1 * quantidade1+valor2*quantidade2) * (IPI / 100 + 1)
20. Faça um programa que lê o número de um funcionário, seu número de horas trabalhadas e o valor que recebe por hora. O programa deve calcular e mostrar o salário deste funcionário.
21. Faça um programa para efetuar o cálculo do valor de uma prestação em atraso, usando a fórmula: PRESTAÇÃO = VALOR+(VALOR*(TAXA/100)*DIAS).