> Os que sonham de dia são conscientes de muitas coisas que escapam aos que sonham apenas à noite.
>
> Allan Kardec

1 Entendendo os objetos

Vimos até agora como armazenar pequenas partes de informações em variáveis e como manipular estas variáveis pela utilização de operadores ou funções. Mas como representar informações mais complexas, principalmente em jogos? Esta representação é possível por meio da utilização de objetos.

Um objeto é uma extensão do conceito de objeto do mundo real, em que se podem ter coisas tangíveis, um incidente (evento ou ocorrência) ou uma interação (transação ou contrato). A figura 1 ilustra esses conceitos:

Figura 1: Tipos de objetos

1.1 Como visualizar um objeto?

Por exemplo, uma espaçonave é um objeto coeso e encapsulado[1], pois combina qualidades (nível de energia ou quantidade de munição restantes) e habilidades (disparar tiros ou levantar o escudo, por exemplo). A figura 2 exemplifica as qualidades e habilidades de uma espaçonave.

Pode-se imaginar um objeto como algo que guarde dentro de si os dados ou informações sobre sua estrutura (seus atributos) e possui um comportamento definido pelas suas operações (seus métodos). Várias linguagens de programação permitem representar e fazer uso dos objetos. Nestas linguagens, uma qualidade é chamada de *atributo* e uma habilidade é chamada de *método*.

A representação de uma espaçonave é genérica, que nós chamamos de *classe*, e a partir dele é possível representar várias espaçonaves distintas (objetos que são manipulados em programação). Na figura 2, a *energia* e a *munição* são os **atributos** da espaçonave e estão declaradas como *int*. *dispararTiro()* e *levantarEscudo()* são os **métodos**.

A representação de objetos em C++ se dá pela declaração de variáveis do mesmo tipo da classe. Neste caso, a variável contém um objeto que representa aquela classe.

[1] **Encapsulação** nada mais é do que esconder detalhes da implementação revelando somente uma pequena interface de uso.

Figura 2: Exemplo de objeto.

```
1  //declaração de variáveis para representar os objetos
2
3  Espaconave minhaNave, inimigo;
```

É possível verificar o conteúdo um atributo ou executar a chamada de um método pela utilização do nome do objeto (variável) separado por um ponto (.) e o nome do atributo/método que se deseja acessar.

```
1  if( minhaNave.energia > 10) // acessando um atributo do objeto
2  {
3     minhaNave.levantarEscudo(); //executando o método
4  }
5  minhaNave.dispararTiro( Inimigo ); //disparando um tiro no inimigo
```

Lembre-se: sempre de associar o termo *atributo* como uma *qualidade* que o objeto tem e um *método* como uma **habilidade** deste objeto.

1.2 Afinal o que é uma classe?

Uma classe é uma coleção de objetos que podem ser descritos por um conjunto básico de atributos e possuem operações semelhantes. A figura 3 ilustra o conceito de classes, de onde é possível partir de uma classe *veículo* (generalização) para diversos tipos de veículos (especialização):

2 Utilizando os objetos de *string*

Uma *string* é perfeito para manipulação de caracteres para um jogo de *puzzle* ou simplesmente para mostrar as histórias dos nossos jogos. As *strings* que utilizamos até o momento são objetos com vários atributos e métodos

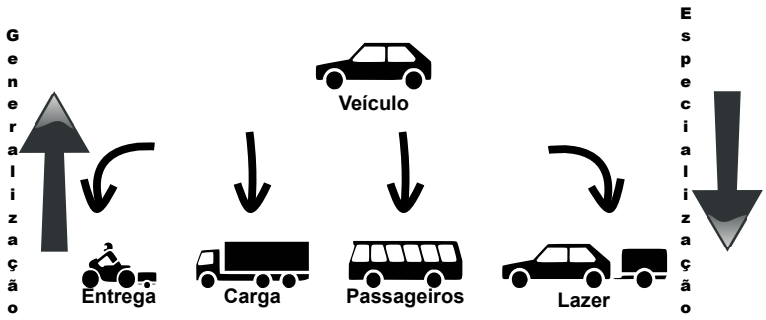

Figura 3: Generalização e especialização de classes.

próprios. Métodos/atributos para obter o tamanho da *string*, somar *strings*, retirar partes da *string*, etc. O programa 1 demonstra a utilização de alguns métodos de um objeto *string*.

Programa 1: Vários testes com a string.

```
1  // trabalhando com strings
2  // programa_001.cpp
3  #include "biblaureano.h"
4
5  int main()
6  {
7      //várias formas de se declarar uma string com um valor inicial
8      string palavra01 = "Fim";
9      string palavra02("Jogo");
10     string palavra03(3,'!'); //repete 3 vezes o mesmo caracter
11
12     string frase = palavra01 + " de " + palavra02 + palavra03;
13
14     cout << "A frase é:" << frase << endl;
15
16     cout << "A frase tem o tamanho de:" << frase.size()
17          << " caracteres." << endl;
18
19     cout << "Modificando a primeira letra da frase" << endl;
20     frase[0] = 'L';
21
22     cout << "A nova frase é:" << frase << endl;
23
24     //imprimindo um caracter por vez
25     for( int posicao=0; posicao < frase.size(); ++posicao)
26     {
27         cout << "Caracter na posição " << posicao <<
28                 " é:[" << frase[posicao] << "]" << endl;
29     }
30
31     //imprimindo a posição de onde começa "de";
32     cout << "A sequência 'de' começa na posição:" << frase.find("de") << endl;
33
34     //verificando se existe a palavra 'game' na frase
35     if( frase.find("game") == string::npos)
36     {
37         cout << "'game' não está na frase." << endl;
38     }
39
40     //apagando parte da string
41     frase.erase(4,5); //apaga a partir da posição 4, 5 caracteres
42     cout << "Nova frase:" << frase << endl;
43
44     frase.erase(4); //apaga a partir da posição 4 até o final
45     cout << "Nova frase:" << frase << endl;
46
47     frase.erase(); //apaga todo o conteúdo da frase
```

```
48      cout << "Nova frase:" << frase << endl;
49
50      //verifica se a frase está vazia
51      if( frase.empty())
52      {
53          cout << "A frase não existe mais!" << endl;
54      }
55      cout << "Game Over" << endl;
56      return 0;
57  }
```

2.1 Criando objetos de strings

A primeira parte do programa, logo após o *main()*, é justamente para declarar varias variáveis do tipo *string*. Lembrando que agora estas variáveis são objetos que *representam* uma *string*.

```
1       //várias formas de se declarar uma string com um valor inicial
2       string palavra01 = "Fim";
3       string palavra02("Jogo");
4       string palavra03(3,'!'); //repete 3 vezes o mesmo caractere
```

Na primeira declaração (linha 2), estamos declarando um objeto e atribuindo o valor "Fim"(como se fosse uma variável qualquer). O resultado é um objeto (*palavra01*) com o conteúdo "Fim".

Na sequência (linha 3), criamos o objeto *palavra02* passando o conteúdo inicial ("Jogo") entre parênteses. O resultado é um objeto com o conteúdo "Jogo". E finalmente (linha 4), o objeto (*palavra03*) é criado passando um número (3) e um caractere ('!'), que indica o caractere deve ser repetido (no caso 3 vezes), o resultado é um objeto com o conteúdo "!!!".

Lembre-se destas formas de declaração de objetos (atribuindo com o sinal = ou passando valores entre parênteses, inclusive valores diferentes). Futuramente, estaremos vendo outros conceitos relacionados a orientação a objetos onde estes conhecimentos serão úteis.

2.2 Concatenando objetos strings

Na sequência, é criado um novo objeto string (*frase*), pela concatenação dos primeiros objetos strings:

```
1   string frase = palavra01 + " de " + palavra02 + palavra03;
```

O resultado é uma nova string com o conteúdo "Fim de Jogo". Repare que o operador +, que antes era utilizado apenas com números, permite a *soma* de strings. Isto ocorre, pois o operador + foi *sobrecarregado* para que funcione com strings.

A *sobrecarga* é outra característica da orientação a objetos é a possibilidade de se definir ou alterar o significado das operações existentes na linguagem, para que seja possível realizar operações sobre objetos construídos em um programa.

2.3 Pegando o tamanho - usando a habilidade *size()*

Agora estamos utilizando pela primeira vez uma habilidade de um objeto, no caso, a habilidade do objeto retornar o seu próprio tamanho.

```
1 cout << "A frase tem o tamanho de:" << frase.size() << " caracteres." << endl;
```

Uma habilidade de um objeto também é chamada de método ou *função membro* do objeto.

O método *size()* retorna o tamanho da string, contando espaços em brancos e qualquer outro caractere que esteja na string em questão.

2.4 Acessando a posição de uma string

Uma *string* armazena uma sequência de caracteres (*char*). É possível acessar individualmente cada valor *char* da *string*. Para isto, basta indexar a posição da string que se deseja acessar informando a posição da *string* entre [ ] (colchetes).

```
1     cout << "Modificando a primeira letra da frase" << endl;
2
3     frase[0] = 'L';
4
5     cout << "A nova frase é:" << frase << endl;
```

Neste exemplo, estamos trocando a primeira letra da *string frase* de 'F' para 'L'.

Como a *string* está trabalhando com uma sequência de valores do tipo *char*, ao se modificar uma posição da *string* deve-se atribuir o novo valor como um *char*, ou seja, entre aspas simples.

Ao contrário do que sugere a lógica, a primeira posição é indicado pelo valor 0, a segunda posição pelo valor 1 e assim sucessivamente. Ou seja, uma *string* com *N* caracteres é indexada da posição 0 até a posição *N*-1.

2.5 *Caminhando* na *string*

Utilizando a estrutura de repetição *for*, já sabendo que a posição inicial de uma *string* é 0 e que o método *size()* retorna o tamanho total da *string*, é possível acessar todas as posições da *string*.

```
1    //imprimindo um caractere por vez
2    for( int posicao=0; posicao < frase.size(); ++posicao)
3    {
4        cout << "Caracter na posição " << posicao << " é:[" << frase[posicao] << "]" << endl;
5    }
```

Durante cada passagem (iteração), é impresso um caractere (*char*) da *string frase*.

Uma *string* com *N* caracteres é indexada da posição 0 até a posição *N*-1, por isto a condição de parada do for é *posicao* < *frase.size()*

2.6 Procurando posições dentro de uma string - usando o método *find()*

O método *find()* permite localizar, na string, onde começa uma determinada sequência de caracteres:

```
1    //imprimindo a posição de onde começa "de";
2    cout << "A sequência 'de' começa na posição:" << frase.find("de") << endl;
```

A posição retornada é a da primeira ocorrência dentro da string. E se a ocorrência não existir dentro da string ? É só comparar o resultado do método *find()* com a constante *string::npos*.

```
1    //verificando se existe a palavra 'game' na frase
2    if( frase.find("game") == string::npos)
3    {
4        cout << "'game não está na frase." << endl;
5    }
```

O método *find()* permite que seja informado a posição inicial da pesquisa na string. Assim, é possível implementar um programa que procure todas as ocorrências de uma string em outra string, basta informar a posição inicial da pesquisa (programa 2).

Programa 2: Procurando várias sequências numa string.

```
// trabalhando com strings
// programa_002.cpp
#include "biblaureano.h"

int main()
{

    string frase = "Fim de de de jogo!!!";
    int posicao = -1; //começo com um valor -1..
    do
    {
        //descarta-se a última posição pesquisada
        posicao = frase.find("de", posicao+1); //..para aqui começar na posição inicial (0)
        if( posicao != string::npos )
        {
            cout << "'de' encontrado na posição " << posicao << endl;
        }
    }
    while( posicao != string::npos);
    cout << "Game Over" << endl;
    return 0;
}
```

2.7 Apagando trechos da string - utilizando *erase()*

O método *erase()* remove uma substring de um objeto string. Basta informar a posição de início e a posição final que se deseja eliminar na string:

```
    //apagando parte da string
    frase.erase(4,5); //apaga a partir da posição 4, 5 caracteres
    cout << "Nova frase:" << frase << endl;

    frase.erase(4); //apaga a partir da posição 4 até o final
    cout << "Nova frase:" << frase << endl;

    frase.erase(); //apaga todo o conteúdo da frase
    cout << "Nova frase:" << frase << endl;
```

Repare que uma chamada a *erase()* sem fornecer as posições é o mesmo que atribuir "" para a string (*Frase* = "").

2.8 Verificando a existência de uma string - o uso do método *empty()*

Finalmente, o método *empty* retorna um valor *bool*, *true* se a string estiver vazia e *false* caso contrário.

```
    //verifica se a frase está vazia
    if( frase.empty())
    {
        cout << "A frase não existe mais!" << endl;
    }
```

3 Brincando com imagens - outra classe da nossa biblioteca.

A nossa biblioteca implementa uma classe, chamada *Imagem*, que utiliza vários métodos para ler um arquivo texto com uma imagem *ASCII*, mostrar a imagem na tela, manipular as coordenadas e ainda verificar a colisão entre 2 imagens. Veja o programa 3:

Programa 3: Brincando com navios.

```
// trabalhando com imagens ascii
// programa_003.cpp
#include "biblaureano.h"

int main()
{
   //pega as imagens dos arquivos
   string n1 = retornaConteudoArquivo("navio1.txt")
   string n2 = retornaConteudoArquivo("navio2.txt")

   // 2 variáveis do tipo imagem, similar a ter 2 várias do tipo string
   Imagem navio1(n1,10,10);
   Imagem navio2(n2,70,10);

   //muda as cores da imagem
   navio1.mudaCor(RED, BLUE);
   navio2.mudaCor(GREEN);

   //verifica se uma imagem colidiu com outra
   while( !navio1.colisao(navio2))
   {
      navio1.imprime();
      navio2.imprime();
      espera(20);
      navio1.limpa();
      navio2.limpa();
      navio1.incrementaX();
      navio2.decrementaX();
   }

   int x1 = 10;
   int x2 = 70;
   int y1 = 3;
   int y2 = 15;
   while( !navio1.colisao(navio2, x1, y1, x2, y2))
   {
      navio1.imprime(x1,y1);
      navio2.imprime(x2,y2);
      espera(50);
      navio1.limpa();
      navio2.limpa();
      x1+=5;
      x2-=5;
      ++y1;
      --y2;
   }
   cout << "Game Over" << endl;
   return 0;
}
```

3.1 Entendendo o programa

Inicialmente criamos as 2 variáveis do tipo *Imagem* passando as strings que contém as imagens em formato *ASCII* e as coordenadas iniciais da imagem:

```
   ...
   //pega as imagens dos arquivos
```

```
    string n1 = retornaConteudoArquivo("navio1.txt")
    string n2 = retornaConteudoArquivo("navio2.txt")

    // 2 variáveis do tipo imagem, similar a ter 2 várias do tipo string
    Imagem navio1(n1,10,10);
    Imagem navio2(n2,70,10);

    //muda as cores da imagem
    navio1.mudaCor(RED, BLUE);
    navio2.mudaCor(GREEN);
    ...
```

Na sequência, é verificado se uma imagem colidiu com a outra. Para fazer o teste é utilizado o método *colisao* da Imagem:

```
    ...
    while( !navio1.colisao(navio2))
    {
        navio1.imprime();
        navio2.imprime();
        espera(20);
        navio1.limpa();
        navio2.limpa();
        navio1.incrementaX();
        navio2.decrementaX();
    }
    ...
```

Os métodos *incrementaX()* e *decrementaX()* não verificam se os limites da tela estão sendo respeitados, cabe a você verificar os limites adequados de acordo com sua imagem e lógica do seu jogo, para tal você deve utilizar os métodos *getX()* e *getY()* para conhecer as coordenadas atuais da imagem:

```
    ...
    if( navio1.getY() < 1 )
    {
        navio1.incrementaY();
    }
    ...
```

Na sequência criamos variáveis externas (*x1*, *y1*, *x2* e *y2*) para manipular as imagens já existente com novas coordenadas:

```
    ...
        int x1 = 10;
    int x2 = 70;
    int y1 = 3;
    int y2 = 15;
    while( !navio1.colisao(navio2, x1, y1, x2, y2))
    {
        navio1.imprime(x1,y1);
        navio2.imprime(x2,y2);
        espera(50);
        navio1.limpa();
        navio2.limpa();
        x1+=5;
        x2-=5;
        ++y1;
        --y2;
    }
    ...
```

A classe imagem contém vários métodos para utilizarmos em nossos jogos, neste momento você estará apto a utilizar os seguintes métodos da classe:

```
1  void imprime(int px, int py); //imprime a imagem nas coordenadas informadas
2  void imprime(); //imprime a imagem nas coordenadas atuais
3  void limpa(int px, int py); //limpa a imagem nas coordenadas informadas
4  void limpa(); //limpa a imagem nas coordenadas atuais
5  void mudaCor(COR pCor); //muda cor da frente
6  void mudaCor(COR pCor, COR pCorFundo); //muda cor frente e fundo
7  bool colisao( Imagem i); //verifica se uma imagem colidiu na outra
8  bool colisao( Imagem i, int x1, int y1, int x2, int y2 ); //verifica se uma imagem colidiu na outra
       nas coordenas informadas
9  int getX(); //retorna o valor atual de x
10 int getY(); //retorna o valor atual de y
11 void setX(int px); //seta o valor x  na imagem
12 void setY(int py); //seta o valor y  na imagem
13 int incrementaX(); //incrementa o valor de x na imagem
14 int incrementaY(); //incrementa o valor de y na imagem
15 int decrementaX(); //decrementa o valor de y na imagem
16 int decrementaY(); //decrementa o valor de y na imagem
```

4 Exercícios

1. É possível utilizar o operador += com strings ?
2. Diga o que o método *length()* faz em uma string.
3. Faça uma pesquisa e relacione os métodos disponíveis de uma string, apresente códigos de exemplo para cada método e explique.
4. Utilizando os métodos *find()* e *erase()* faça um programa para retirar de uma frase fornecida pelo usuário todas as repetições de uma determinada sequência informada pelo usuário. Por exemplo, o usuário informou a frase *Eu gosto de abacaxi e abacate.* e a sequência *aba* seu programa deveria mostrar *Eu gosto de caxi e cate.*
5. Desenvolva uma história que possa ser contada através de uma sequência de imagens, você pode se inspirar no site http://www.asciimation.co.nz/