Aula 4

Objectivos: (Continuação da aula anterior.)

Preâmbulo:

Para realizar o desenho de gráficos – como é requerido em alguns problemas do grupo A desta aula – ir-se-á utilizar uma biblioteca gráfica simples criada expressamente para esta Cadeira (existindo versões quer para `Windows` quer para `linux`). Esta biblioteca é composta por duas classes: `SIMPLE_GRAPHICAL_OUTPUT` e `SIMPLE_GRAPHICAL_INPUT`, que se podem utilizar por intermédio de dois objectos globais únicos: "`sgout`" e "`sgin`" (em Windows, o ficheiro de interface pode ser encontrado em:
`c:\Program Files\Qt\include`).
A utilização destas classes é feita seguindo os seguintes passos:

1. Incluir o ficheiro de interface do módulo:

   ```
   #include "simple_graphical_module.h"
   ```

2. Inicializar a janela gráfica:

   ```
   sgout.initialize();
   ```

3. Requerer o desenho de elementos gráficos (texto, pontos, círculos, etc.):

   ```
   sgout.add_text(20,30,orange,"Texto!"); // texto a partir das
                                          // coordenadas (20,30)
                                          // com a cor laranja
   sgout.add_point(7,7,black);
   // (...)
   ```

4. Por forma a manter actualizado o conteúdo da janela gráfica, por vezes é necessário redesenhar alguns elementos gráficos. Isso é feito através do método:

   ```
   sgout.update_window();
   ```

   Este método actualiza a janela gráfica (se necessário) e devolve o controlo do programa ao cliente. Como alternativa pode-se passar o controlo do programa para a janela gráfica até que seja premida alguma tecla ou algum botão do rato nessa janela:

```
event = sgin.event_wait();
```

O valor de "event" identifica o evento ocorrido (`'a'` para a tecla "a", `'1'` para a tecla "1", `mouse_left_button`, para o botão direito do rato, etc.)

Quando a janela é fechada é devolvido o valor: `exit_event`.

Em Windows, por forma a se poder utilizar esta biblioteca no Visual Studio *C++*, torna-se necessário fazer algumas alterações no projecto *Win32 Console Application*. Para tal basta (após o projecto estar criado) correr o programa `pp1cnvprj` e seleccionar o projecto (ficheiro com terminação ".dsp") desejado (durante esta operação o projecto não deve estar aberto pelo Visual Studio *C++*).

Problema A1

Pretende-se fazer um programa para lidar com figuras geométricas. Nesta primeira versão pretende-se apenas a possibilidade de criar algumas figuras, com uma determinada posição no espaço (use um espaço a duas dimensões, embora tentando fazer com que a "interface" pública das classes dependa o mínimo possível disso), e efectuar o seu desenho.

A1.1. Crie (e teste) as classes adequadas para as seguintes figuras: rectângulo, quadrado e triângulo.

A1.2. Implemente uma classe lista de figuras (use a implementação interna da lista que lhe pareça mais prática).

A1.3. Faça um programa que manipule uma lista de figuras, dando a possibilidade de o utilizador criar e acrescentar novas figuras, e que as desenhe a todas.

A1.4. Acrescente as seguintes figuras: polígono, elipse e círculo.

A1.5. Acrescente a possibilidade (método `void mover(int dx,int dy);`) de mover as figuras no espaço.

Problema A2

Prosseguindo o problema A1 anterior, pretende-se agora acrescentar a possibilidade de cada figura devolver as coordenadas do seu centro. Considere

que o centro de uma qualquer figura corresponde ao seu centro de massa. Para um polígono o seu centro de massa é calculado pela seguinte fórmula:

$$C_m = \frac{\sum_{i=1}^{n} d_i}{n}$$

,em que d_i é vector da posição de cada vértice, e n o número de vértices.

A2.1. Defina para uma qualquer figura um método que determine a distância entre essa figura e uma outra qualquer (sendo esta definida como a distância entre os respectivos centros).

```
class FIGURA
{
public:
  (...)

  double distancia(FIGURA *outra_figura);

  (...)
};
```

Problema B1

B1.1. Construa a classe `CPessoa` contendo três variáveis-membros: idade, nome e apelido de uma pessoa. Implemente as funções necessárias para consultar/modificar estes atributos. Implemente a função `GetInfo()` que devolve o nome e o apelido da pessoa.

B1.2. Construa a classe `CProfessor` derivada da classe `CPessoa`. A classe `CProfessor` deve ter um atributo adicional: `m_sArea` – que indica a área principal de investigação do docente. Redefina a função `GetInfo()` para que esta devolva para além do nome e apelido, o título "Prof.". Não reescreva a função `GetInfo()` completamente, aproveite da função existente na classe base.

B1.3. Construa a classe `CAluno` derivada da classe `CPessoa`. A classe `CAluno` deve ter um atributo adicional: `unsigned m_nNumMec` – o número mecanográfico. Redefina a função `GetInfo()` para que esta devolva para além do nome e apelido, também o número mecanográfico do aluno. Não reescreva a função `GetInfo()` completamente, aproveite a função existente na classe base.

B1.4. Construa a classe `CAlunoPosGraduacao` derivada da classe `CAluno`. A classe `CAlunoPosGraduacao` deve ter um atributo adicional que será um apontador para o respectivo orientador: `CProfessor* m_pProfResp`. Redefina a função `GetInfo()` para que esta devolva para além do nome, apelido e número mecanográfico do aluno, o nome, apelido e título do orientador. Não reescreva a função `GetInfo()` completamente, aproveite da função existente na classe base.

B1.5. Crie na função main alguns objectos das classes `CPessoa`, `CAluno`, `CAlunoPosGraduacao` e `CProfessor`. Chame a função `GetInfo` para cada um destes. Repare na ordem em que são chamados os construtores/destrutores para as classes bases e derivadas.

B1.6. Construa a classe `CDisciplina` contendo os membros seguintes:

```
CProfessor* m_pProf;          // professor responsável
unsigned m_nNumeroAlunos;     // número de alunos inscritos
CAluno** m_arAlunos;          // array de ponteiros para os alunos
char* m_sNome;                // nome da disciplina
```

Imagine que numa disciplina podem inscrever-se os alunos de licenciatura e os de pós-graduação e que o número de alunos é ilimitado. Inicialmente não

se sabe quantos alunos se vão inscrever. Assegure que o "array" `m_arAlunos` cresce dinamicamente com a ajuda dos operadores `new` e `delete`.

B1.7. Implemente a seguinte função `main`. A função global:
`void InfoDisciplina(CDisciplina& disc)`
deve imprimir toda a informação sobre a disciplina, incluindo o seu nome, informação sobre o professor responsável, número de alunos inscritos e os dados de todos os alunos (obtidos com a função `GetInfo()`).

```
int main(int argc, char* argv[])
{
  CAluno a1 ("Antonio", "Santos", 22, 12345);
  CAluno a2 ("Maria", "Gonçalvez", 18, 12456);
  CAluno a3 ("Rita", "Bastos", 18, 12789);
  CAluno a4 ("José", "Melo", 21, 13456);
  CAluno a5 ("Manuel", "Brito", 23, 45789);
  CAluno a6 ("Nuno", "Santiago", 24, 45689);
  CAluno a7 ("Nelson", "Rocha", 25, 45189);

  CProfessor p1 ("Francisco", "Cardoso", 35, "Programação");
  CDisciplina disc1 ("Programação Orientada por Objectos", &p1);
  disc1.InscreverAluno(&a1);
  disc1.InscreverAluno(&a2);
  disc1.InscreverAluno(&a6);
  disc1.InscreverAluno(&a7);

  CProfessor p2 ("Jorge", "Matos", 45, "Sistemas Digitais");
  CDisciplina disc2 ("Sistemas Digitais Avançados", &p2);

  disc2.InscreverAluno(&a1);
  disc2.InscreverAluno(&a2);
  disc2.InscreverAluno(&a4);
  disc2.InscreverAluno(&a5);
  disc2.InscreverAluno(&a6);

  InfoDisciplina (disc1);
  InfoDisciplina (disc2);

  return 0;
}
```

Problema B2

O código seguinte está correcto? Caso não esteja, indique todos os erros.

```
class A
{
  int a1;

public:
  A (int f = 0, int s = 0, int t = 0) : a1(f), a2(s), a3(t) { };
  virtual ~A() {};

  int a2;

protected:
  int a3;
};
```

```
class B : public A
{
  int b1, b2, b3;

public:
  B() : A(), b1(a1), b2(a2), b3(a3) {  };
  virtual ~B() {};
};
```

```
int main(int argc, char* argv[])
{
  B* b = new B();
  return 0;
}
```