Paradigmas de Programação I

Guia das Aulas Práticas

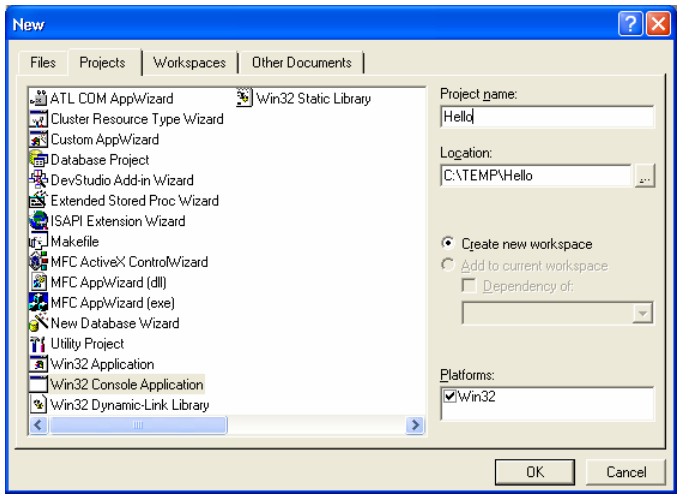
Andreia Melo
Arnaldo Oliveira

Ano lectivo 2002-2003

O ambiente de trabalho utilizado nas aulas práticas de Paradigmas de Programação I é o Microsoft Visual C++ 6.0 a executar sobre o sistema operativo Microsoft Windows. Para se familiarizar com este ambiente e como introdução ao paradigma de orientação por objectos, propõem-se de seguida alguns exercícios.

Exercício 1

1.1 Crie um projecto novo (menu *File* | *New*...). Seleccione o separador *Projects* e escolha a opção *Win32 Console Application*. Crie o seu projecto no directório local `C:\Temp` e dê-lhe o nome *hello*.

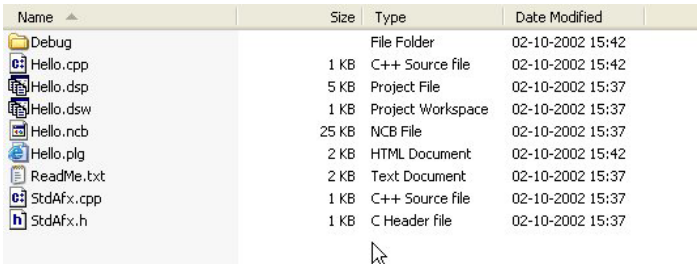


Na próxima janela terá de escolher o tipo de aplicação que vai criar. Escolha a opção *A "Hello, World!" application*. Ao criar o projecto desta forma é também criado um *workspace* com o mesmo nome. Um *workspace* pode conter vários projectos, eventualmente de tipos diferentes (*EXE*-consola/windows, *DLL*-biblioteca dinâmica, *LIB*-biblioteca estática), etc.).

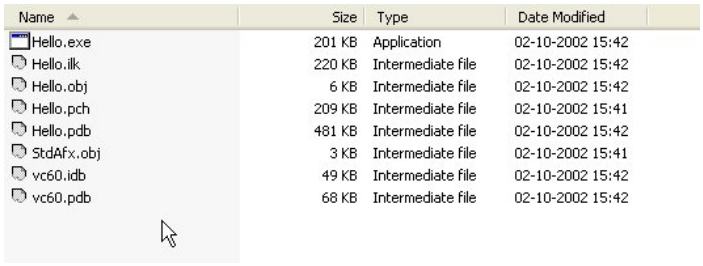
Durante a criação de um projecto são gerados vários ficheiros:

- Ficheiro de *workspace* com extensão `dsw` (`hello.dsw` neste caso);
- Ficheiro de projecto com extensão `dsp` (`hello.dsp` neste caso);
- Ficheiro(s) de interface (*headers*) com extensão `.h`;
- Ficheiro(s) de implementação com extensão `.cpp`.

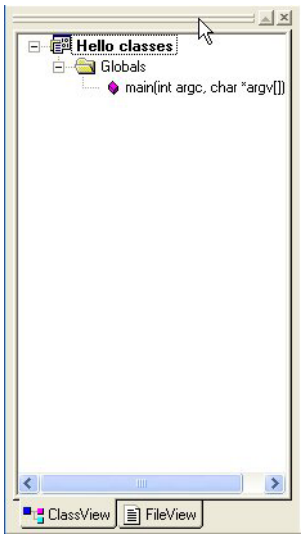
A função main do programa é normalmente colocada no ficheiro `<nome do projecto>.cpp` (`hello.cpp` neste caso). Os ficheiros `StdAfx.h` e `StdAfx.cpp` são usados pelo compilador na geração de *headers* pré-compilados. Tudo o que o utilizador precisa é incluir, no ficheiro `StdAfx.h`, os ficheiros de interface das bibliotecas da linguagem que necessita para o projecto (ex. `string.h`, `iostream.h`, etc.).



Sempre que desejar abrir um *workspace* e os projectos nele contidos deverá escolher a opção *File* | *Open Workspace...* e selecionar o respectivo ficheiro. Para fechar um *workspace* escolha a opção *File* | *Close Workspace*. Para criar o ficheiro executável (compilar e fazer o *linking*) seleccione a opção *Build* | *Build Hello.exe* (F7) ou *Build* | *Rebuild all*. Para executar o programa escolha a opção *Build* | *Execute Hello.exe* (CTRL+F5).
Durante a compilação do projecto são gerados os ficheiros objecto (`*.obj`), o ficheiro executável (`*.exe`), bem como outros ficheiros intermédios. Todos eles ficam armazenados no directório *Debug* pertencente ao directório do projecto. O conteúdo deste directório pode ser apagado manualmente ou usando a opção *Build* | *Clean*.



A janela *Workspace* possui dois separadores: *Class View* e *File View*. No primeiro podemos ver a árvore de classes, de funções e variáveis globais do nosso projecto. Neste exemplo foi gerada apenas uma função global `main`. No segundo separador temos uma lista de todos os ficheiros pertencentes ao projecto. Um directório *Header Files* armazena os módulos `*.h` e um directório *Source Files* armazena os módulos `*.cpp`. Tal como já foi dito, a função `main` encontra-se no módulo `hello.cpp`.



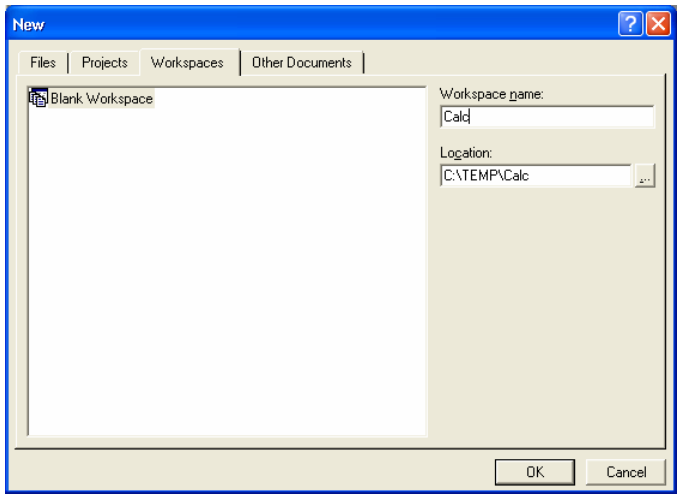
1.2 Substitua a invocação da função `printf` pelo operador << do C++ que em conjunto com o objecto `cout` permite escrever informação na *stream* standard de saída (`stdout`). Para tal deve incluir o ficheiro `iostream.h` no seu projecto.

```
printf("Hello World!\n");     ->     cout << "Hello World!" << endl;
```

1.3 Com funcionalidade equivalente à da função `scanf` do C, existe em C++ o operador >> que em conjunto com o objecto `cin` permite ler informação da *stream* standard de entrada (`stdin`). Utilize-o no seu programa para efectuar a leitura de um valor para uma variável do tipo `int`, fazendo de seguida a impressão no ecrã do respectivo valor.

Exercício 2

2.1 Vamos agora construir um programa que dados dois operandos reais, do tipo `double`, e uma operação (`+, -, *, /`), determine o resultado e o escreva no ecrã. Para tal, crie um *workspace* vazio chamado *Calc*. Seguidamente, adicione um projecto do tipo *Win32 Console Application* chamado *DoubleCalc* (para facilitar, escolha a opção *A simple application*).



O projecto depois de criado possui três ficheiros (`DoubleCalc.cpp; StdAfx.h; StdAfx.cpp`). O ficheiro `DoubleCalc.cpp` contém a função `main` que deve ser substituída pelo seguinte código de forma a implementar a funcionalidade pretendida.

```
int main(int argc, char* argv[])
{
	double operand1, operand2, result;
	char operation;

	cout << "Operand 1 : ";
	cin >> operand1;
	cout << "Operation (+-*/) : ";
	cin >> operation;
	cout << "Operand 2 : ";
	cin >> operand2;

	switch(operation)
	{
		case '+':
		{
			result = operand1 + operand2;
			break;
		}

		case '-':
		{
			result = operand1 - operand2;
			break;
		}

		case '*':
		{
			result = operand1 * operand2;
			break;
		}

		case '/':
		{
			result = operand1 / operand2;
			break;
```

```
                }

                default:
                {
                        cout << "Invalid operation" << endl;
                        return -1;
                }
        }

        cout << "Result = " << result << endl;
        return 0;
}
```

Introduza este programa, execute-o e faça o seu teste com diferentes valores e operações.

2.2 Suponhamos agora que queremos efectuar as mesmas operações mas sobre números complexos em vez de números reais. Assumindo que existia um tipo de dados capaz de representar números complexos, a solução mais conveniente seria substituir o tipo de dados reais (`double`), usado na declaração das variáveis, por um tipo que represente números complexos (ex. `CComplex`). Neste caso a função `main` passaria a ser escrita da seguinte forma:

```
int main(int argc, char* argv[])
{
        CComplex operand1, operand2, result;

        // O restante código permaneceria inalterado
}
```

É importante notar que a única coisa que é alterada é declaração das variáveis para armazenar os operandos e o resultado. No entanto, o tipo `CComplex` não é um tipo de dados predefinido e os operadores `+, -, *, /, >>, <<` não podem ser utilizados directamente. Contudo, em C++ a implementação desta solução é possível desde que se defina o tipo `Ccomplex` e se implemente as funções e os operadores necessários. A isto chama-se abstracção de dados. Para chegarmos a essa solução vamos considerar ainda um passo intermédio, definindo uma classe e algumas funções que operam sobre números complexos.

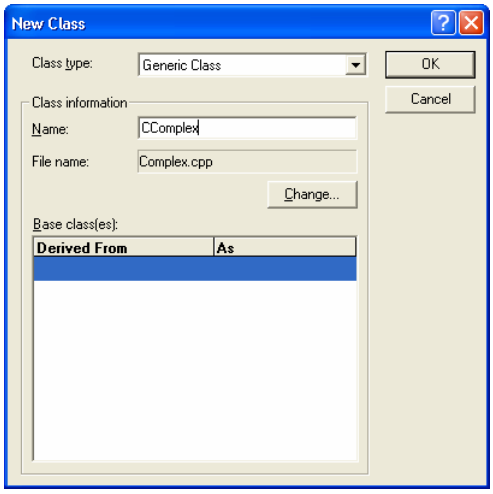
2.3 Acrescente ao *workspace Calc* um novo projecto, chamado *BasicCalc*, efectuando os mesmos passos do exercício 1 e escolhendo a opção *Simple Application*. Tal como anteriormente, pode verificar que foi gerado o módulo (`BasicCalc.cpp`) com a função `main`. O objectivo deste exercício é substituir os operadores usados no programa anterior por funções com funcionalidade equivalente mas que operam sobre números complexos. Assim, a função `main` passaria a ser escrita da seguinte maneira:

```
#include "stdafx.h"
#include "Complex.h"

int main(int argc, char* argv[])
{
        CComplex operand1, operand2, result;
        char operation;

        cout << "Operand 1 : ";
        Input(operand1);
        cout << "Operation (+-*/) : ";
        cin >> operation;
        cout << "Operand 2 : ";
        Input(operand2);

        switch(operation)
        {
                case '+':
                {
```

```
                        result = Add(operand1, operand2);
                        break;
                }

                // Introduzir aqui o restante código

                default:
                {
                        cout << "Invalid operation" << endl;
                        return -1;
                }
        }

        cout << "Result = ";
        Output(result);
        return 0;
}
```

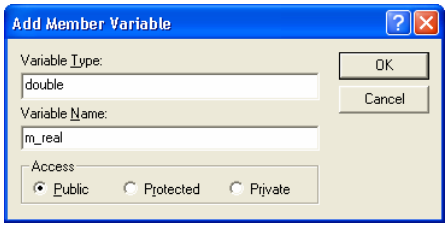
Criação da Classe

O próximo passo é criar uma classe que represente um número complexo e inseri-la no projecto. Utilize o menu *popup* (com o botão direito do rato, clique sobre o nome do projecto no separador *Class View* da janela *Workspace*). Escolha neste menu a opção *New Class...* e dê-lhe o nome `CComplex`. Quando se cria esta classe, são criados dois ficheiros: o de interface (`Complex.h`) e o de implementação (`Complex.cpp`).



Adição de Atributos

Os atributos da classe `CComplex` são a parte real e a parte imaginária do número complexo, ambos números reais. Adicione estes atributos editando o ficheiro `Complex.h` ou utilizando o menu *popup* (com o botão direito do rato clique sobre o nome da classe no separador *Class View* da janela *Workspace*). Neste menu deve escolher a opção *Add Member Variable...*



Repita este procedimento para os dois atributos `m_real` e `m_imag` (ambos do tipo `double`).

```
// Interface – Ficheiro Complex.h
class CComplex
{
// Constuctors, destructors
public:
        CComplex();
        virtual ~CComplex();

// Atributes
public:
        double m_real;
        double m_imag;
};
```

O Construtor

O protótipo e implementação do construtor por defeito (constructor sem parâmetros) e do destrutor são criados automaticamente quando se cria a classe. Inicialize a parte real e a parte imaginária do número complexo a zero no construtor da classe.

```
// Implementação – Ficheiro Complex.cpp
CComplex::CComplex()
{
        m_real = 0;
        m_imag = 0;
}

CComplex::~CComplex()
{
}
```

Funções Globais

As função global `Input` apresentada no código abaixo permite introduzir um número complexo. A função `Add` calcula a soma de dois números complexos.

```
// Interface – Ficheiro Complex.h
class CComplex
{
// Constuctors, destructors
public:
        CComplex();
        virtual ~CComplex();

// Atributes
public:
        double m_real;
        double m imag;
};

void Input(CComplex& c);

CComplex Add(const CComplex& c1, const CComplex& c2);
```

```
//Implementação – Ficheiro Complex.cpp
CComplex::CComplex()
{
        m_real = 0;
        m imag = 0;
}

CComplex::~CComplex()
{
}

void Input(CComplex& c)
{
        cin >> c.m real >> c.m imag;
}
```

```
CComplex Add(const CComplex& c1, const CComplex& c2)
{
	CComplex result;

	result.m_real = c1.m_real + c2.m_real;
	result.m_imag = c1.m_imag + c2.m_imag;

	return result;
}
```

Construa, baseando-se na função `Add`, as funções `Sub`, `Mult` e `Div`, também globais, para subtrair, multiplicar e dividir dois complexos, respectivamente. Implemente também a função `Output`, para escrever o valor de um complexo na *stream* standard de saída. Complete a função `main` do projecto *BasicCalc* de forma a utilizar estas funções, sendo os argumentos os dois números complexos `c1` e `c2` lidos do `cin`.

2.3 Embora já estejamos preparados para efectuar todas as operações sobre números complexos, esta não é a forma mais conveniente de o fazer. O último passo para atingirmos o objectivo deste exercício é a redefinição dos operadores `+`, `-`, `*`, `/`, `>>` e `<<` correspondentes às funções `Add`, `Sub`, `Mult`, `Div`, `Input` e `Output`, respectivamente. Desta forma passa a ser possível escrever a função `main` como foi apresentada acima, ou seja, em tudo idêntica à que operava sobre números reais. Para isso, vamos acrescentar um novo projecto, chamado *ComplexCalc*, ao *workspace*. Mais uma vez é conveniente escolher a opção *A simple application*. Como exemplo é fornecida a redefinição do operador +, para somar dois complexos, e do operador >> para ler o seu valor.

```
// Interface - Ficheiro Complex.h
class CComplex
{
// Constuctors, destructors
public:
	CComplex();
	virtual ~CComplex();

// Methods
public:
	friend istream& operator>>(istream& is, CComplex& c);

	CComplex operator+(const CComplex& c);

// Atributes
public:
	double m_real;
	double m_imag;
};
```

```
// Implementação - Ficheiro Complex.cpp
CComplex::CComplex()
{
	m_real = 0;
	m_imag = 0;
}

CComplex::~CComplex()
{
}

istream& operator>>(istream& is, CComplex& c)
{
	is >> c.m_real >> c.m_imag;
	return is;
}

CComplex CComplex::operator+(const CComplex& c)
{
	CComplex result;

	result.m_real = m_real + c.m_real;
```

```
        result.m_imag = m_imag + c.m_imag;

        return result;
}
```

Tendo por base estes operadores, redefina os operadores `-`, `*`, `/` e `<<` de forma a poder utilizar a seguinte função `main` contida no ficheiro `ComplexCalc.cpp`:

```
int main(int argc, char* argv[])
{
        CComplex operand1, operand2, result;
        char operation;

        cout << "Operand 1 : ";
        cin >> operand1;
        cout << "Operation (+-*/) : ";
        cin >> operation;
        cout << "Operand 2 : ";
        cin >> operand2;

        switch(operation)
        {
                case '+':
                {
                        result = operand1 + operand2;
                        break;
                }

                case '-':
                {
                        result = operand1 - operand2;
                        break;
                }

                case '*':
                {
                        result = operand1 * operand2;
                        break;
                }

                case '/':
                {
                        result = operand1 / operand2;
                        break;
                }

                default:
                {
                        cout << "Invalid operation" << endl;
                        return -1;
                }
        }

        cout << "Result = " << result << endl;
        return 0;
}
```

2.4 Pretende-se agora expandir a funcionalidade da classe anterior de forma a calcular o módulo (`Mod`) e o ângulo (`Ang`) das respectivas coordenadas polares de um complexo, e a determinar o seu conjugado (`Conj`). O primeiro método é apresentado como exemplo.
Para adicionar métodos à classe `CComplex` edite directamente os ficheiros `Complex.h` e `Complex.cpp` ou escolha a opção *Add Member Function...* do menu *popup*, tal como fez para adicionar um atributo.

```
// Interface - Ficheiro Complex.h
class CComplex
{
// Constuctors, destructors
public:
        // ...
```

```
// Methods
public:
        double Mod();

// Atributes
public:
        // ...
};
```

```
// Implementação – Ficheiro Complex.cpp
double CComplex::Mod()
{
        return sqrt(pow(m real, 2) + pow(m imag, 2));
}
```

Defina os métodos `Ang` e `Conj` e estenda o programa *ComplexCalc* de forma a utilizar estes métodos. Use os caracteres 'm', 'a' e 'c' para identificar as operações. Tenha em atenção que as operações anteriores (`+, -, *, /`) são binárias enquanto as operações `Mod, Ang` e `Conj` são unárias, pelo que não precisam do segundo operando.

2.5 Pretende-se agora acrescentar à classe `CComplex` o método `int GetQuadrant();` que devolve um número inteiro correspondente ao quadrante a que pertence o complexo (1, 2 3 ou 4). No entanto, em vez de se fazer este cálculo cada vez que o método é invocado, pretende-se acrescentar à classe um atributo chamado `m_quadrant` que indique o respectivo quadrante. Este atributo deve ser mantido coerente com as coordenadas do complexo, isto é, sempre que a parte real e/ou a parte imaginária do complexo se altera, este atributo tem de ser recalculado. O método `int GetQuadrant();` limita-se a devolver o valor do atributo `m_quadrant`. No entanto, existe um problema: como os atributos `m_real` e `m_imag` foram até agora definidos sempre como públicos, é possível alterar cada uma das componentes do número complexo sem que se altere o atributo relativo ao quadrante, o que faz com que seja possível corromper, do exterior, a integridade do objecto. Por este motivo, os atributos de uma classe devem em geral ser privados ou protegidos e o seu valor deve ser alterado de forma controlada através de métodos públicos que mantêm a integridade interna do objecto. Este mecanismo permite também separar o interface da implementação da classe e designa-se por encapsulamento da informação. Assim, altere a visibilidade dos atributos `m_real` e `m_imag` para `private` e acrescente o atributo `m_quadrant`, também `private`. Defina o método `void Set(double real, double imag);` usado para alterar o valor do complexo e os métodos `double Real();` e `double Imag();` que devolvem cada uma das suas componentes. Defina o método `void SetQuadrant();` que deve ser `private` ou `protected`. Este método deve ser invocado dentro dos construtores e dos métodos da classe que alterem o valor das coordenadas do complexo. Por último, acrescente um constructor com o protótipo `CComplex(double real, double imag);` de forma a ser possível iniciar o complexo com valores arbitrários durante a criação do objecto.

Exercício 3

3.1 Escreva um programa que leia no máximo 10 números complexos e armazene num *array* apenas aqueles que pertencem ao 1º ou 4º quadrantes.

3.2 Seguidamente, o programa deve ordenar os complexos por ordem crescente de quadrante e, dentro do mesmo quadrante por ordem crescente da parte real. Por último deve escrever os números complexos ordenados no ecrã. Exemplo:

```
1: 4 + 1i; quadrante 1
2: 7 + 2i; quadrante 1
3: 1 - 4i; quadrante 4
4: 5 - 2i; quadrante 4
```