TEMPLATE em C++

A declaração template especifica um conjunto de classes ou funções parametrizadas. Os compiladores mais recentes de C++ permitem que os programadores definam funções genéricas ou classes baseados em argumentos que desconhecem o tipo. Esse tipo só será definido quando forem usadas por entidades específicas.

A declaração template tem o seguinte aspecto:

template <lista de parâmetros template**>** , a lista será separada por vírgulas e não pode ser vazia

EXEMPLO de função template :

```
template <class T>
void troca(T &a, T &b)
{
        T temp=a;
        a=b;
        b=temp;
}
```

Neste exemplo a função troca ( ), troca os dois objectos a e b entre si, podendo estes ser instâncias de qualquer classe .O tipo genérico aqui usado é T. Os parâmetros actuais da função troca ( ), podem ser inteiros, ou valores reais, ou objectos do tipo pessoa ou círculo ou ...
A aplicação desta função poderia ser feita do seguinte modo:

```
main()
{
int     a=3,
        b=10;

double x=3.99,
        y=5.72;

circulo circ1(10,75,34,"Azul"),
        circ2( 5,88,99,"Verde");

        troca(a,b);
        troca(x,y);
        troca(circ1,circ2);     // nesta classe deveríamos ter definido o construtor cópia e
                                a sobrecarga do operador de atribuição uma vez que tal é
                                exigido pelo função troca ( ).
}
```

Quando o compilador encontra a utilização da função template troca ( ) é que é gerado o código. Como consequência disto a definição de uma função template deve ser feita num ficheiro cabeçalho.
No exemplo acima, são criadas 3 funções, uma para manipular inteiros ,outra para double e outra para círculos.

No exemplo apresentado só havia um parâmetro T mas poderiam ser mais e nesse caso apareceriam separados por vírgulas na lista entre < >

Exemplo:

```
 template <class T1, classT2>
void funçãoA ( T1 &x, T2 &y)
{
....      ....    .....
}
```

Uma classe template comporta-se como um mecanismo para a geração automática de instância de classe de tipo específico.

EXEMPLO de classe template

1.

```
template <class T>
class celula
{
        private:
        T produto;
        int valor;
        public:
        celula ();
        T & getproduto();
        void setproduto(const T &x);

};
```

2.

```
::: ::: ::: :::
template <class T>
T & celula<T>::getproduto()
{
        if(valor)
                return produto;
}

template <class T>
void celula<T>::setproduto(const T & x)
{
        valor++;
```

```
        produto=x;    // a classe com que vai instanciar T terá que ter definido o operador
        de atribuição
}

... ... ...
```

3.
Utilização da classe template celula

```
#include <iostream.h>
#include "celula.h"
#include "circulo.h"

void main()
{
        celula <int> A;
        int x;
        celula <circulo> B;
        circulo circ(11,12,88,"Lilas");
        x=35;
        A.setproduto(x);
        B.setproduto(circ);
        cout<<A.getproduto()<<endl;
        cout<<B.getproduto()<<endl;  //teremos que fazer a sobrecarga do operador<<
        para que seja possível fazer a escrita de um objecto do tipo circulo.
}
```

4.
Sobrecarga do operador <<
Se aparece cout<<objecto teremos que sobrecarregar o operador << de modo a aceitar um operando do tipo ostream e outro operando do tipo a que o objecto pertence.

Solução1.
Construir uma função que faz essa sobrecarga e considerá-la como friend da classe a que o objecto pertence.
No caso de o objecto ser da classe circulo teríamos:

```
ostream& operator <<(ostream & ostr, const circulo & c)
{
        ostr<<"Coordenada x="<<c.getcoordx()<<'\n'<<"Coordenada
        y="<<c.getcoordy()<<'\n' << "Raio="<<c.getraio()<<'\n'<<"Cor="<<c.getcor()
        <<'\n';
        return ostr;
}
```

Supomos que temos definidos os métodos públicos getcoordx(), ... que devolvem o valor dos respectivos membros de dados

Conclusão: teríamos que ter tantas definições do operador quantas as classes que a nossa aplicação manipulava. Essas definições não eram membros das classes, mas simplesmente funções.

Solução 2
Esta aproximação será a que utilizaremos e consiste em construir uma função template que faz a sobrecarga do operador <<

```
template <class T>>
ostream & operator <<(ostream &ostr, const T &x)
{
        x.escrever(ostr);
        return ostr;
}
```

Na classe circulo teríamos que incluir o método escrever ( ) cujo protótipo seria:

```
void escrever(ostream &ostr) const;
```

Este método terá que ser const uma vez que o objecto, x, que executa o método é constante,

e definir-se-ia:

```
void circulo::escrever(ostream &ostr) const
{
        ostr<<"Coordenada x="<<c.coordx<<'\n'<<"Coordenada
        y="<<c.coordy<<'\n' << "Raio="<<c.raio<<'\n';
}
```

Conclusão: com esta solução só temos que definir uma vez o operador << como uma função template que por sua vez invoca o método escrever ( ). Este método tem que ser membro da classe a que pertence o objecto que pretendemos fazer cout.