Templates

Каждый гениальный человек смотрит на мир под другим углом от своих собратьев.
Havelock Ellis




Шаблоны функций и шаблоны классов позволяют программистам определить, с одного сегмента кода, весь спектр сопутствующих (перегрузки) функции называется функция-шаблон специализации или весь спектр связанных классов называется класс-шаблон специализации. Эта техника называется обобщенное программирование.

Мы могли бы написать один шаблон функции для массива функции сортировки, то есть C + + генерировать отдельную функцию-шаблон специализации

что будет сортировать Int массивов,
поплавок массивов,
строка массивов
и так далее.

Шаблоны функций

Функция шаблоны специальных функций, которые могут работать с универсальными типами.

шаблон < идентификатор класса > объявление функции;
шаблон < TypeName идентификатор > объявление функции;

Единственное различие между двумя прототипами является использование либо класс ключевое слово или ключевое слово TypeName.

```
template <class myType>
myType GetMax (myType a, myType b)
{
        return (a>b?a:b);
}
```

```
template< typename T >
void printArray( const T *array, int count )
{
  for ( int i = 0; i < count; i++ )
    cout << array[ i ] << " ";
  cout << endl;
}
```

```
#include <iostream>
using namespace std;

template <class T>
T GetMax (T a, T b)
{
  T result;
  result = (a>b)? a : b;
  return (result);
}

int main ()
{
  int i=5, j=6, k;
  long l=10, m=5, n;
  k=GetMax<int>(i,j);
  n=GetMax<long>(l,m);
  cout << k << endl;
  cout << n << endl;
  return 0;
}
```

```
6
10
```

```
#include <iostream>
using namespace std;

template <class T>
T GetMax (T a, T b)
{
  return (a>b?a:b);
}

int main ()
{
  int i=5, j=6, k;
  long l=10, m=5, n;
  k=GetMax(i,j);
  n=GetMax(l,m);
  cout << k << endl;
  cout << n << endl;
  return 0;
}
```

```
6
10
```

Что делать, если у вас есть 2 выбора. Открытая или закрытая

Мы также можем определить шаблоны функций, которые принимают больше чем один тип параметров, просто указав дополнительные параметры шаблона между угловыми скобками.

```
template <class T, class U>
T GetMin (T a, U b) {
return (a<b?a:b);
}
```

В этом случае шаблон функции GetMin () принимает два параметра различных типов и возвращает объект того же типа, что и первый параметр (T), которая передается

```
int i,j;
long l;
i = GetMin<int,long> (j,l);
```

or : i = GetMin (j,l);

даже если J и L имеют различные типы, так как компилятор может определить подходящий экземпляр в любом случае.

Шаблоны классов

Класс шаблоны называются параметризованных типов, потому что они требуют одного или нескольких параметров типа, чтобы указать, как настроить "универсальный класс" шаблон, чтобы сформировать класс-шаблон специализации.

```
template <class T>
class mypair
{
    T values [2];
  public:
    mypair (T first, T second)
    {
      values[0]=first;
      values[1]=second;
    }
};
```

```
mypair<int> myobject (115, 36);
```

```
mypair<double> myfloats (3.0, 2.18);
```

```
#include <iostream>
using namespace std;

template <class T>
class mypair {
    T a, b;
  public:
    mypair (T first, T second)
      {a=first; b=second;}
    T getmax ();
};

template <class T>
T mypair<T>::getmax ()
{
  T retval;
  retval = a>b? a : b;
  return retval;
}

int main () {
  mypair <int> myobject (100, 75);
  cout << myobject.getmax();
  return 0;
}
```

```
100
```

Как Grade Efficiently

Creating Class Template Stack< T >
(Создание класса стека шаблона)

```
#ifndef STACK_H
#define STACK_H

template< typename T >
class Stack
{
 public:
    Stack( int = 10 );
    ~Stack() ;
    bool push( const T& );
    bool pop( T& );
    bool isEmpty() const ;
    bool isFull() const;
 private:
    int size;
    int top;
    T *stackPtr;
};
#endif
```

```
#include "Stack.h"

int main()
{
   Stack< double > doubleStack( 5 );
   double doubleValue = 1.1;
   cout << "Pushing elements onto doubleStack\n";
    while ( doubleStack.push( doubleValue ) )
    {
          cout << doubleValue << ' ';
          doubleValue += 1.1;
    }

    while ( doubleStack.pop( doubleValue ) )
           cout << doubleValue << ' ';

    Stack< int > intStack;
    return 0;
}
```

Заметки о Шаблоны и наследование

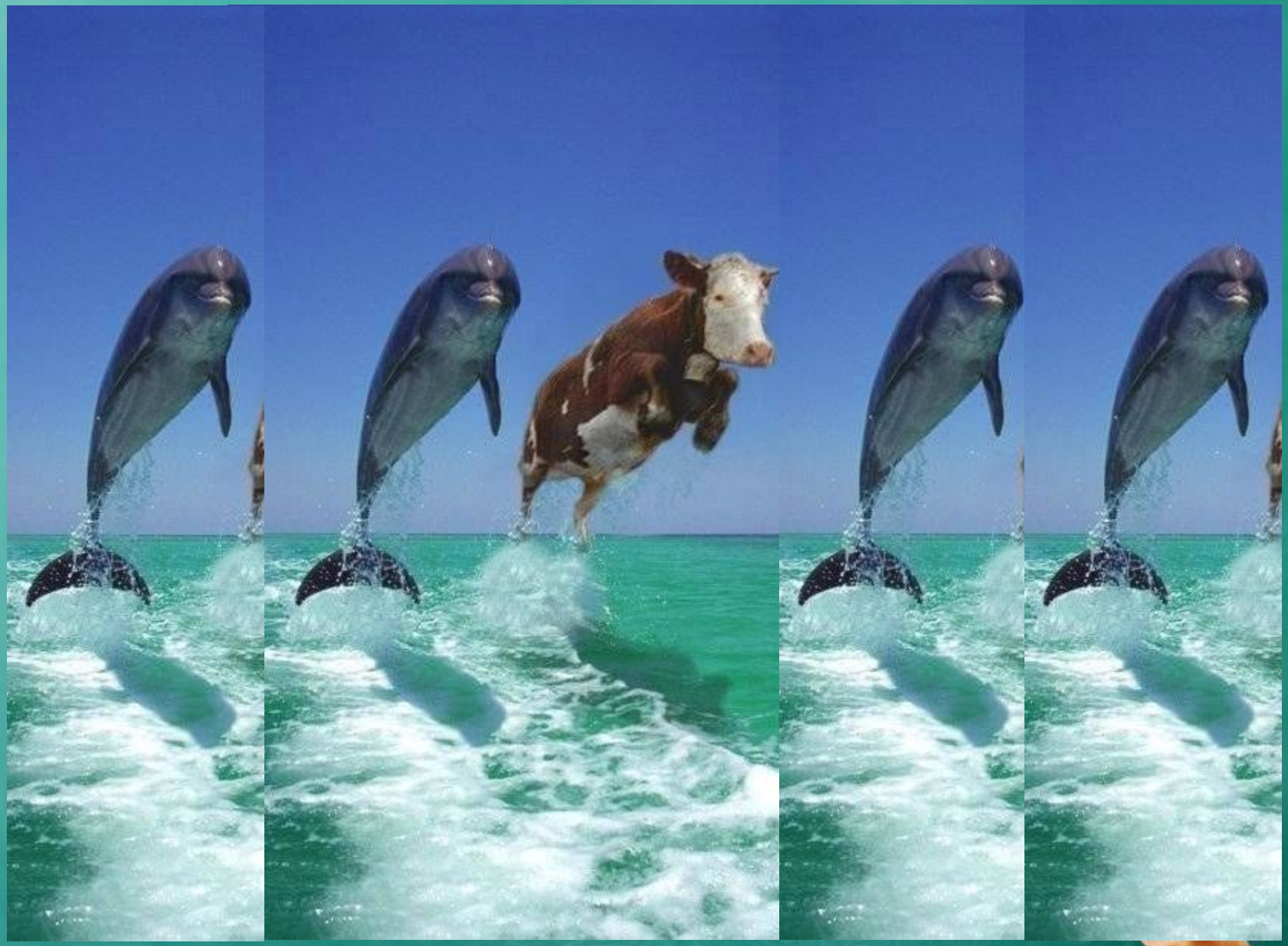
Шаблоны и наследование связаны несколькими способами:

Шаблон класса может быть получена из класса-шаблона специализации.

Шаблон класса может быть получена из класса нешаблонными.

Класс-шаблон специализации могут быть получены из класса-шаблона специализации.

Класс нешаблонными могут быть получены из класса специализации шаблона.



Найти разницу

Шаблон специализации

Если мы хотим, чтобы определить различные реализации для шаблона, когда определенный тип передается в качестве параметра шаблона, мы можем объявить специализации этого шаблона

Давайте предположим, что у нас есть очень простой класс MyContainer, который может хранить один элемент любого типа, и что он имеет только одну функцию-член называется увеличение, что повышает его ценность.

Но мы видим, что, когда он сохраняет элемент типа символов было бы удобнее иметь совершенно разные реализации с прописной функция, поэтому мы решили объявить специализации шаблона класса для данного типа

```
template <class T>
class mycontainer

template <> class mycontainer <char> { ... };
```

```
// class template:
template <class T>
class mycontainer {
    T element;
  public:
    mycontainer (T arg) {element=arg;}
    T increase () {return ++element;}
};
```

```
// class template specialization:
template <>
class mycontainer <char> {
    char element;
  public:
    mycontainer (char arg) {element=arg;}
    char uppercase ()
    {
      if ((element>='a')&&(element<='z'))
      element+='A'-'a';
      return element;
    }
};
```

```
int main ()
{
  mycontainer<int> myint (7);
  mycontainer<char> mychar ('j');
  cout << myint.increase() << endl;
  cout << mychar.uppercase() << endl;
  return 0;
}
```

For which template will
Mycontainer<int, char> myA(5,'a');

Номера типа параметров для шаблонов

Кроме того, аргументы шаблона, которые предшествовали класса или TypeName ключевые слова, которые представляют типы, шаблоны могут также иметь регулярную типизированные параметры, аналогичные тем, которые содержатся в функциях.

```
template <class T, int N>
class mysequence {
    T memblock [N];
  public:
    void setmember (int x, T value);
    T getmember (int x);
};

template <class T, int N>
void mysequence<T,N>::setmember (int x, T
value)
{
  memblock[x]=value;
}

template <class T, int N>
T mysequence<T,N>::getmember (int x) {
  return memblock[x];
}
```

```
int main () {
  mysequence <int,5> myints;
  mysequence <double,5> myfloats;
  myints.setmember (0,100);
  myfloats.setmember (3,3.1416);
  cout << myints.getmember(0) << '\n';
  cout << myfloats.getmember(3) <<
'\n';
  return 0;
}
```

Шаблоны и нескольких файлов проектов

С точки зрения компилятора, шаблоны не являются нормальными функциями или классами. Они составлены на спрос, а это означает, что код шаблона функции не компилируются до экземпляра с конкретными аргументами шаблона не требуется.
В тот момент, когда экземпляр требует, компилятор генерирует функции специально для тех аргументов шаблона.

Поскольку шаблоны компилируются при необходимости, это заставляет ограничение для нескольких файлов проектов: реализация (определение) шаблон класса или функции должны быть в тот же файл, как его заявлении. Это означает, что мы не можем отделить интерфейс в отдельный файл заголовка, и что мы должны включать в себя как интерфейса и реализации в любой файл, который использует шаблоны.

Поскольку код не генерируется, пока экземпляр шаблона, когда это требуется, компиляторы готовы разрешить включение несколько раз одного и того же файла шаблона с обеих деклараций и определений, содержащихся в проекте, не создавая связь ошибки.

Назарларыңызға Рахмет =)