Subject: Re: NEW: generic Toupel grouper
Posted by dolik.rce on Fri, 13 Aug 2010 23:40:14 GMT
View Forum Message <> Reply to Message

So, after criticizing kohaits proposals I felt obligated to show my own idea about how Touples should be implemented... Here is the result: #include <Core/Core.h>
using namespace Upp;

```
String AsString(Nuller n){return "";}

template <int N,class A,class B,class C> struct TTypes         {typedef Nuller Type;};
template <class A,class B,class C>       struct TTypes<0,A,B,C> {typedef A Type;};
template <class A,class B,class C>       struct TTypes<1,A,B,C> {typedef B Type;};
template <class A,class B,class C>       struct TTypes<2,A,B,C> {typedef C Type;};

template<class A, class B,class C> struct Touple;

template<int N,class A,class B,class C> struct Retriever{
 static Nuller Get(Touple<A,B,C>& t){return Null;};
};
template<class A,class B,class C> struct Retriever<0,A,B,C>{
 static typename TTypes<0,A,B,C>::Type Get(Touple<A,B,C>& t){return t.a;};
};
template<class A,class B,class C> struct Retriever<1,A,B,C>{
 static typename TTypes<1,A,B,C>::Type Get(Touple<A,B,C>& t){return t.b;};
};
template<class A,class B,class C> struct Retriever<2,A,B,C>{
 static typename TTypes<2,A,B,C>::Type Get(Touple<A,B,C>& t){return t.c;};
};

template<class A, class B=Nuller,class C=Nuller>
struct Touple{
 A a;
 B b;
 C c;
 Touple(){};
 Touple(A a):a(a){};
 Touple(A a,B b):a(a),b(b){};
 Touple(A a,B b,C c):a(a),b(b),c(c){};

 template<int N>
 typename TTypes<N,A,B,C>::Type Get(){
  return Retriever<N,A,B,C>::Get(*this);
 };
 template<class T>
 Touple& operator=(const T& t){
  a=t.a; b=t.b; c=t.c;
 }
```

```
 int GetCount()const{
  for(int i=2; i>0; i--){
   if((*this)[i]!=Value(Null)) return i+1;
  }
  return 1;
 }
 Value operator[](int i)const{
  if     (i==0) return Value(a);
  else if(i==1) return Value(b);
  else if(i==2) return Value(c);
  else ASSERT_(false,"index out of bounds");
 }
 String ToString()const{
  String s=AsString(c);
  s=AsString(b)+(s.GetLength()>0?",":"")+s;
  s=AsString(a)+(s.GetLength()>0?",":"")+s;
  return "{"+s+"}";
 }
};

template<class A>
Touple<A> Solo(const A& a){
 return Touple<A>(a);
};

template<class A,class B>
Touple<A,B> Duo(const A& a,const B& b){
 return Touple<A,B>(a,b);
};

template<class A,class B,class C>
Touple<A,B,C> Trio(const A& a,const B& b,const C& c){
 return Touple<A,B,C>(a,b,c);
};

CONSOLE_APP_MAIN{
 Touple<double,const char*> s;
 Touple<int,String> t;
 Touple<int,String,double> u;

 t.a=1; t.b="test";
 DUMP(t.Get<0>()); DUMP(t.Get<1>());
 for(int i=0; i<t.GetCount(); i++){
  LOG(i<<": "<<t[i]);
 }
 s=t;                          DUMP(s);
 t=Duo(1,String("hello world"));  DUMP(t);
 u=t;                          DUMP(u);
```

```
 u.c=3.2;                    DUMP(u);
 // <double>=<triple> fails to compile:
 // t=Triple(1,String("dsd"),3);  DUMP(t);
}
```

The main difference is that there is no specific type for two,three, etc. values, but rather a single Touple class that can be used universally. The implementation above allows 1,2 or 3 elements, but can be easily extended. The main idea is that smaller touple can be assigned into bigger (extra elements are Null), while bigger into smaller triggers compilation errors. The functions Solo,Duo and Trio are supposed to save you some typing by generating the touples based on the arguments types. The only thing I found missing in Core was AsString(Nuller) which should be no problem to add.

The ideas from my previous post are included. I am aware that touple.Get<0>() doesn't have any significant syntactic value, since it is the same as touple.a, but it might help the readability a bit. On the other hand, the operator[] is quite important, since it allows iterating through the Touple. TEven though it returns Values, together with GetCount() (returning number of elements from 0 to last non-Null) should be quite powerfull tool.

What do you think?

Honza

PS: The Solo,Duo and Trio names were chosen just because there already is One and Single. Otherwise I would prefer Single, Double, etc.
