Subject: the container type UPP::Link
Posted by piotr5 on Sun, 28 Apr 2013 11:10:36 GMT
View Forum Message <> Reply to Message

strange container, really. it stores N links to T where T is expected to be derived from Link. you could imagine it as N double-linked lists with each list-element being associated to a subset of those lists, and with each of those lists being connected back into a full loop. Interestingly only the 1st list gets destructed in the destructor of LinkOwner, guess this is the list that must contain all members. useful it is to represent different permutations of the same data. in uppsrc it is used only once, /home/p/upp/uppsrc/CtrlCore/CtrlTimer.cpp(22) for storing TimeEvent , and there N is set to 1 -- guess in future std::list could be used instead.

there actually is a bug in its implementation: it should betemplate <class T, int N = 1>
class LinkOwner : public T {
public:
 ~LinkOwner()   { T::DeleteList(); }
};
otherwise compiling template<typename T, int N=1>
class LinkElement : public Link<LinkElement<T,N>,N>
{
 public:
 T val;
};
LinkOwner<LinkElement<int,2> > delme; (note that N is 2 here, with N=1 it would compile OK) will

/home/p/upp/uppsrc/Core/Other.h:305:7:   required from here

  T *link_prev[N];
          ^
/home/p/upp/uppsrc/Core/Other.h:273:64: error: within this context
  void Unlink(int i = 0)   { link_next[i]->link_prev[i] = link_prev[i]; link_prev[i]->link_next[i] = link_next[i];
                                                                ^

  T *link_next[N];
          ^
/home/p/upp/uppsrc/Core/Other.h:273:107: error: within this context
  void Unlink(int i = 0)   { link_next[i]->link_prev[i] = link_prev[i]; link_prev[i]->link_next[i] = link_next[i];
guess I must use a different LinkOwner then...
Edit: in case it wasn't clear, Link is connected in full circle back to itself, so iterating will eventually

return to where you started...

Subject: Re: the container type UPP::Link
Posted by mirek on Fri, 10 May 2013 06:21:17 GMT
View Forum Message <> Reply to Message

piotr5 wrote on Sun, 28 April 2013 07:10strange container, really. it stores N links to T where T is expected to be derived from Link. you could imagine it as N double-linked lists with each list-element being associated to a subset of those lists, and with each of those lists being connected back into a full loop. Interestingly only the 1st list gets destructed in the destructor of LinkOwner, guess this is the list that must contain all members. useful it is to represent different permutations of the same data. in uppsrc it is used only once, /home/p/upp/uppsrc/CtrlCore/CtrlTimer.cpp(22) for storing TimeEvent , and there N is set to 1 -- guess in future std::list could be used instead.

there actually is a bug in its implementation: it should betemplate <class T, int N = 1>
class LinkOwner : public T {
public:
 ~LinkOwner()   { T::DeleteList(); }
};
otherwise compiling template<typename T, int N=1>
class LinkElement : public Link<LinkElement<T,N>,N>
{
 public:
 T val;
};
LinkOwner<LinkElement<int,2> > delme; (note that N is 2 here, with N=1 it would compile OK) will result in[code]

template <class T>
class LinkElement : public Link<LinkElement<T>, 2> {
public:
 T val;
};

LinkOwner< LinkElement<int>, 2 > delme;

is intended use and it works with current LinkOwner (and does not with proposed patch).

Other than that, I guess I can agree that Link is not the most important class in U++... Original idea was to simplify creation of multilinked lists, but the need for them diminished over time.

Mirek