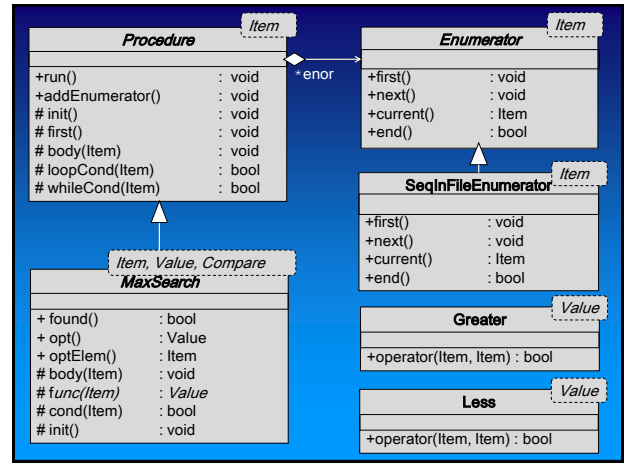
Objektum elvű alkalmazások fejlesztése

Programozási tételek újrafelhasználása 2.

Készítette: Gregorics Tibor



3.Feladat

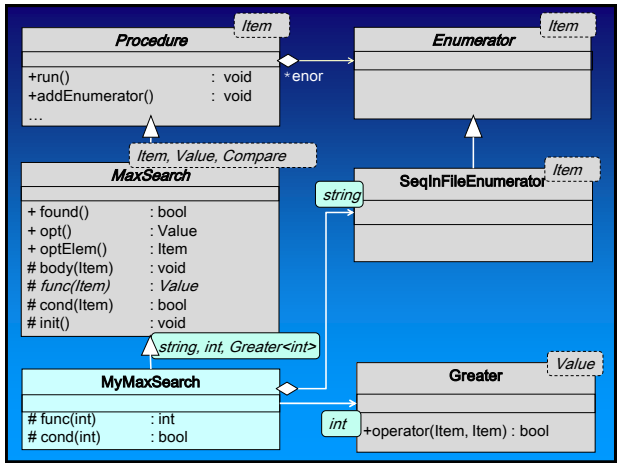
Adott egy szöveges fájlbeli szöveg, ahol a szavakat szóközök, tabulátor-jelek, sorvége-jelek illetve a fájlvége-jel határolja. Melyik a leghosszabb olyan szó, amelyikben a legnagyobb kódú karakter a 'w' betű?

Megoldás

„Melyik a leghosszabb adott tulajdonságú szó?"

A feladat visszavezethető a feltételes maximum keresésre

1. Vizsgált elemek (Item): szavak (string)
2. Összehasonlítandó értékek (Value): szavak hossza (int)
3. Függvény (Func): előállítja egy szó hosszát (size)
4. Összehasonlítás (Compare): „nagyobb" reláció (Greater)
5. Feltétel (Cond): a szó legnagyobb kódú karaktere a 'w'
6. Felsoroló: szöveges állomány szavainak felsorolója



Implementáció

```
class MyMaxSearch : public MaxSearch<string, int>{
protected:
   int  func(const string& e) const{ return e.size();}
   bool cond(const string& e) const;
};

int main()
{
   MyMaxSearch pr;
   SeqInFileEnumerator<string> word_enor("text.txt");
   pr.addEnumerator(&word_enor);

   pr.run();

   if (pr.found())
        cout << "A keresett szó " << pr.optElem() << endl;
   else cout << "Nincs keresett tulajdonságú szó\n";
   return 0;
}
```

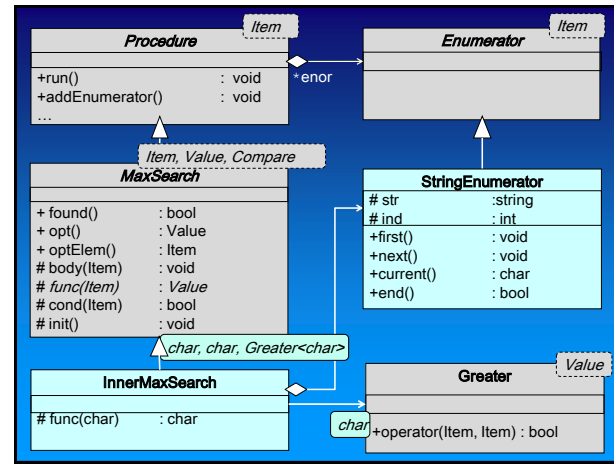
main.cpp

Részfeladat megoldása

„Egy szó legnagyobb kódú karaktere a ‘w’?”

A feladat visszavezethető a maximum kiválasztásra

1. Vizsgált elemek (Item): karakterek (char)
2. Összehasonlítandó értékek (Value): karakterek (char)
3. Függvény (Func): identitás
4. Összehasonlítás (Compare): „nagyobb” reláció (Greater)
5. Feltétel (Cond): igaz
6. Felsoroló: egy sztring karaktereinek felsorolója



Implementáció

```
class StringEnumerator : public Enumerator<char> {
protected:
   string str;
   int ind;
public:
   StringEnumerator(const string& s): str(s){}
   void first()          { ind = 0;}
   void next()           { ++ind;}
   bool end()      const { return ind>=(int)str.size();}
   char Current() const { return str[ind]; }
};

class InnerMaxSearch : public MaxSearch<char>{
protected:
   char  Func(const char& e) const{ return e;}
};

bool MyMaxSearch::Cond(const string& e) const{
   InnerMaxSearch pr;
   StringEnumerator enor(e); pr.AddEnumerator(&enor);
   pr.Run();
   return 'w'==pr.OptElem();
};
```

main.cpp

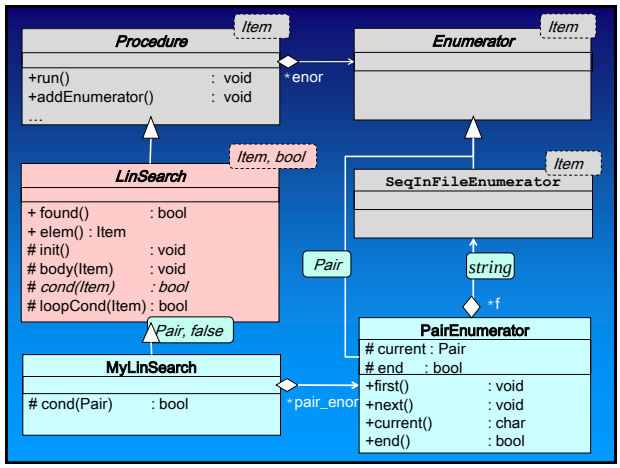
4.Feladat

Egy szöveges állományban neveket soroltak fel ábécé szerint rendezve. Keressük meg az első olyan nevet amelyik legalább ötször szerepel az állományban!

„Melyik az első legalább ötször szereplő név?”

A feladat visszavezethető egy lineáris keresésre

1. Vizsgált elemek (Item): név-darabszám párok
2. Feltétel (Cond): legalább öt darabszámú pár
3. Felsoroló: név-darabszám párok felsorolása, amely a szöveges állománybeli nevek felsorolására épül



Lineáris keresés programozási tétele

```
template < typename Item, bool optimist = false>
class LinSearch : public Procedure<Item> {
protected:
   bool _l;
   Item _elem;

   void init() {_l = optimist;}
   void body(const Item& e) {_l = cond(_elem = e);}
   bool loopCond() const {
      return (optimist?l:!_l) && Procedure<Item>::loopCond();
   }
   virtual bool cond(const Item& e) const = 0;

public:
   bool found()  const { return _l;}
   Item elem()   const { return _elem;}
};
```

linsearch.hpp

Kitérő: Kiválasztás programozási tétele

```
template < typename Item >
class Selection : public Procedure<Item> {
protected:
   void init(){}
   void body(const Item& e) {}
   bool loopCond() const {
        return !cond(Procedure<Item>::enor->current());
   }
   virtual bool cond(const Item& e) const = 0;
public:
   Item result() { return Procedure<Item>::enor->current(); }
};
```

selection.hpp

Implementáció

```
struct Pair{
   string name;
   int    count;
};

class MyLinSearch : public LinSearch<Pair>{
public:
   bool cond(const Pair &e) const { return e.count>=5; }
};

int main()
{
   MyLinSearch pr;
   PairEnumerator t("text.txt");
   pr.addEnumerator(&t);
   pr.run();

   if (pr.found())
      cout << "A " << pr.elem().name << " az első";
   else cout << "Nincs";
   cout << " olyan név, amelyet legalább öten viselnek.\n";
   return 0;
}
```

main.cpp

Absztrakt felsoroló

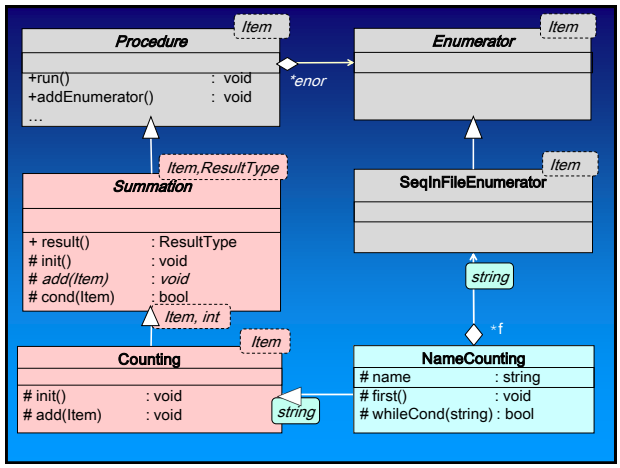
```
class PairEnumerator : public Enumerator<Pair> {
protected:
   SeqInFileEnumerator<string> *_f;
   Pair _current;
   bool _end;
public:
   PairEnumerator(const string& str)
            { _f = new SeqInFileEnumerator<string>(str); }
   ~PairEnumerator(){ delete _f; }
   void first() { _f->first(); next(); }
   void next();
   bool end()      const { return _end;}
   Pair current() const { return _current; }
};
```

main.cpp

Absztrakt olvasó művelet (Next()) által megoldandó részfeladat

„Hányszor szerepel a soron következő név?”

A feladat visszavezethető a számlálásra (speciális összegzés) a szöveges állománybeli szavak felsorolójával. (Ezt adja a SeqInFileEnumerator<string>) A részfeladat megoldásához ezt a felsorolót azonban nem az állomány teljes végigolvasására kell használni, hiszen egyrészt a soron következő név előtt már sok nevet beolvastunk vele, tehát nem kell előre olvasás, másrészt a felsorolást csak addig kell folytatni, amíg az aktuális név nem változik.



Összegzés programozási tétele

```
template < typename Item, typename ResultType = Item >
class Summation : public Procedure<Item>
{
protected:
   ResultType *_result;
   bool _inref;

   Summation(){ _result = new ResultType; _inref = true; }
   Summation(ResultType *r) :_result(r) { _inref = false; }
   void init(){}
   void body(const Item& e) { if(cond(e)) add(e);}

   virtual void add(const Item& e) = 0;
   virtual bool cond(const Item& e) const { return true; }

public:
   ResultType result() { return *_result;}
   ~Summation()        { if(_inref) delete _result;}
};
```

Szokásos felüldefiniálása: *_result := 0

Létező hivatkozás legyen!

Szokásos felüldefiniálása : *_result := *_result + f(e);

summation.hpp

Számlálás programozási tétele

```
template < typename Item >
class Counting : public Summation<Item, int> {
   public:
      Counting():Summation<Item,int>(){}

   protected:
      void init(){
            *Summation<Item,int>::_result = 0;}
      void add(const Item& e) {
            ++*Summation<Item,int>::_result; }
};
```

counting.hpp

Absztrakt olvasó művelet (Next()) által megoldandó részfeladat részletesen

„Hányszor szerepel a soron következő név?”

A feladat visszavezethető a számlálásra

1. Vizsgált elemek (Item): nevek (string)
2. Feltétel (cond): igaz
3. Felsoroló: szöveges állománybeli szavak felsorolása
4. Extra terminálási feltétel (whileCond): amíg a felsorolt név nem változik
5. Extra indulás (first): üres (a soron következő név ismert)

Absztrakt olvasó művelet: Next()

```
class NameCounting : public Counting<string>
{
   protected:
      string _name;

      void first(){}
      bool whileCond(const string& e) const
               { return e == _name; }
   public:
      NameCounting(const string &str):
               Counting<string>(), _name(str){}
};

void PairEnumerator::next(){
   if(_end = _f->end()) return;
   _current.name = _f->current();
   NameCounting pr(_current.name); pr.addEnumerator(_f);

   pr.run();

   _current.count = pr.result();
}
```

main.cpp

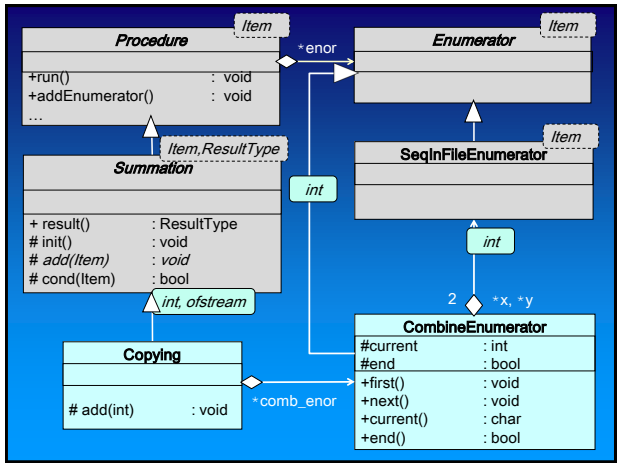
5.Feladat

Adott két szöveges állomány, bennük egész számok szigorúan növekedően rendezve. Gyűjtsük ki az összes számot, de mindegyiket csak egyszer egy harmadik szöveges állományba!

„Futtassuk össze a két állományt (unió)!”

A feladat visszavezethető az összegzésre

1. Vizsgált elemek (Item): egész számok
2. Eredmény(ResultType): egész számok sorozata adatstream-en
3. Tevékenység (Add): egész szám adatstream-be írása
4. Felsoroló: két szekvenciális inputfájl számainak összefésülése



Felsorolt elemek kiírása

```
class Copying : public Summation<int,ofstream >
{
   public:
      Copying(ofstream *f):Summation<int, ofstream >(f){}
   protected:
      void add(const int& e){*_result << e << endl;}
};

int main()
{
   ofstream u("uj.txt");  if(u.fail()) exit(2);
   Copying pr(&u);
   CombineEnumerator comb_enor("1.txt","2.txt");
   pr.AddEnumerator(&comb_enor);

   pr.Run();

   return 0;
}
```

main.cpp

Összefuttató felsoroló

```
class CombineEnumerator : public Enumerator<int>
{
   protected:
      SeqInFileEnumerator<int> *_x;
      SeqInFileEnumerator<int> *_y;
      int _current;
      bool _end;
   public:
      CombineEnumerator(const std::string& str1,
                        const std::string& str2){
         _x = new SeqInFileEnumerator<int>(str1);
         _y = new SeqInFileEnumerator<int>(str2);
      }
      ~CombineEnumerator(){ delete _x; delete _y; }

      void first() { _x->first(); _y->first(); next(); }
      void next();
      bool end()    const { return _end;}
      int  current()const { return _current;}
};
```

main.cpp

Összefuttató felsoroló Next() művelete

```
void CombineEnumerator::next()
{
   if( _end = _x->end() && _y->end() ) return;

   if( _y->end() || !_x->end()
                 && _x->current() < _y->current() ){
      _current = _x->current();
      _x->next();
   }else if( _x->end() || !_y->end()
                 && _x->current() > _y->current() ){
      _current = _y->current();
      _y->next();
   }else if(!_x->end() && !_y->end()
                 && _x->current() == _y->current() ){
      _current = _x->current();
      _x->next(); _y->next();
   }
}
```

main.cpp

Kitérő: Next() művelet csak a közös elemek felsorolásához

```
void CombineEnumerator::next()
{
   bool l = false;
   while(!l)
      if( _end = _x->end() || _y->end() ) return;

      if( _x->current() < _y->current() ){
         _x->next();
      }else if( _x->current() > _y->current() ){
         _y->next();
      }else if( _x->current() == _y->current() ){
         _current = _x->current();
         l = true;
         _x->next(); _y->next();
      }
   }
}
```

main.cpp

Kitérő: Next() művelet csak a közös elemek felsorolásához

```
void CombineEnumerator::next()
{
   if( _end = _x->end() || _y->end() ) return;

   if( _x->Current() < _y->current() ){
      _x->next();
      next();
   }else if( _x->current() > _y->current() ){
      _y->next();
      next();
   }else if( _x->current() == _y->current() ){
      _current = _x->current();
      _x->next(); _y->next();
   }
}
```

main.cpp

Kitérő: Next() művelete az összes elem jelzett felsorolásához

```
Item← struct( Element data , string sign)
void CombineEnumerator::next()
{
   if( _end = _x->end() && _y->end() ) return;

   if( _y->end() || !_x->end()
                 && _x->current() < _y->current() ){
      _current.data = _x->current(); _current.sign = "x";
      _x->next();
   }else if( _x->end() || !_y->end()
                 && _x->current() > _y->current() ){
      _current.data = _y->current(); _current.sign = "y";
      _y->next();
   }else if(!_x->end() && !_y->end()
                 && _x->current() == _y->current() ){
      _current.data = _x->current(); _current.sign = "xy";
      _x->next(); _y->next();
   }
}
```

main.cpp