Feladat

Adott egy szöveges fájlbeli szöveg, ahol a szavakat szóközök, tabulátor-jelek, sorvége-jelek illetve a fájlvége-jel határolja. Melyik a leghosszabb 'W' betűt tartalmazó szó?

Megoldás

Programterv

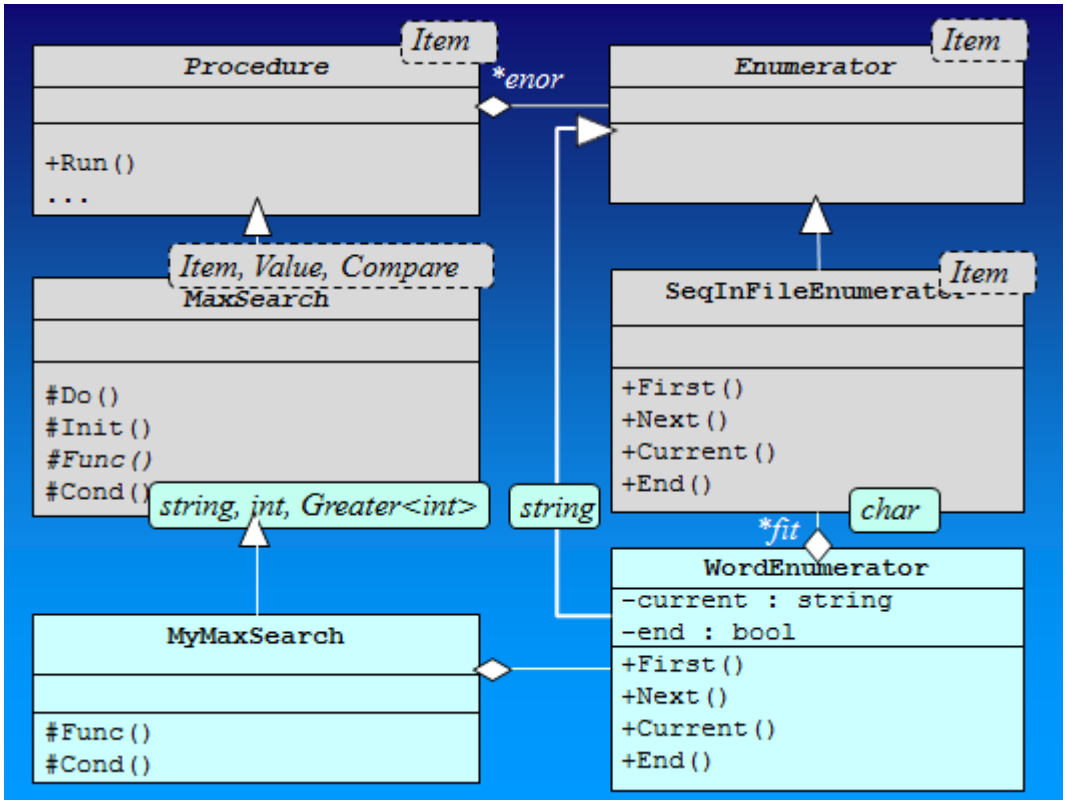
A megoldáshoz érdemes elképzelni egy olyan absztrakt felsorolót, amely a karakterenkénti olvasást elrejti, és a szöveg szavait képes felsorolni úgy, hogy minden szót megjelöl aszerint, hogy van-e benne 'W' betű. A feladatot egy ilyen felsorolóra épített feltételes maximumkereséssel oldjuk meg.

$$A = (f{:}infile(Szó),\ l{:}\mathrm{L},\ szó{:}String) \quad Szó = rec(str : String\ ,\ vanW : \mathrm{L})$$

$$Ef = (\ f{=}f'\ )$$

$$Uf = (l,\ max,\ szó = \max_{\substack{i=1 \\ t_i'.vanW}}^{|t'|} |t_i'.str|\ )$$

A maximum kiválasztást kód-újrafelhasználással valósítjuk meg. Feltételezve, hogy rendelkezésünkre állnak a `Procedure`, `Enumerator`, `SeqInFileEnumerator` és `MaxSearch` osztály-sablonok (lásd előadás), ezért nekünk csak a `MyMaxSearch` és a `WordEnumerator` osztályokat kell definiálni.



A `WordEnumerator` osztály *t* példányát a szöveg karakterenkénti *fit* felsorolója, valamint a felsorolás során éppen érintett aktuális szó (*akt*) és a *vége* logikai érték reprezentálja. A felsorolás műveleteinek implementációját az alábbi táblázat mutatja.

enor(Szó)	*First()*	*Next()*	*End()*	*Current()*
*fit : enor(*K) *akt : Szó* *vége :* L	*első_szót_keres(fit)* *vége := ¬f it.End()* *ha ¬ vége akkor* *előállít(akt, fit)*	*vége := ¬ fit.End()* *ha ¬ vége akkor* *előállít(akt, fit)*	*vége*	*akt*

Részletesebben:

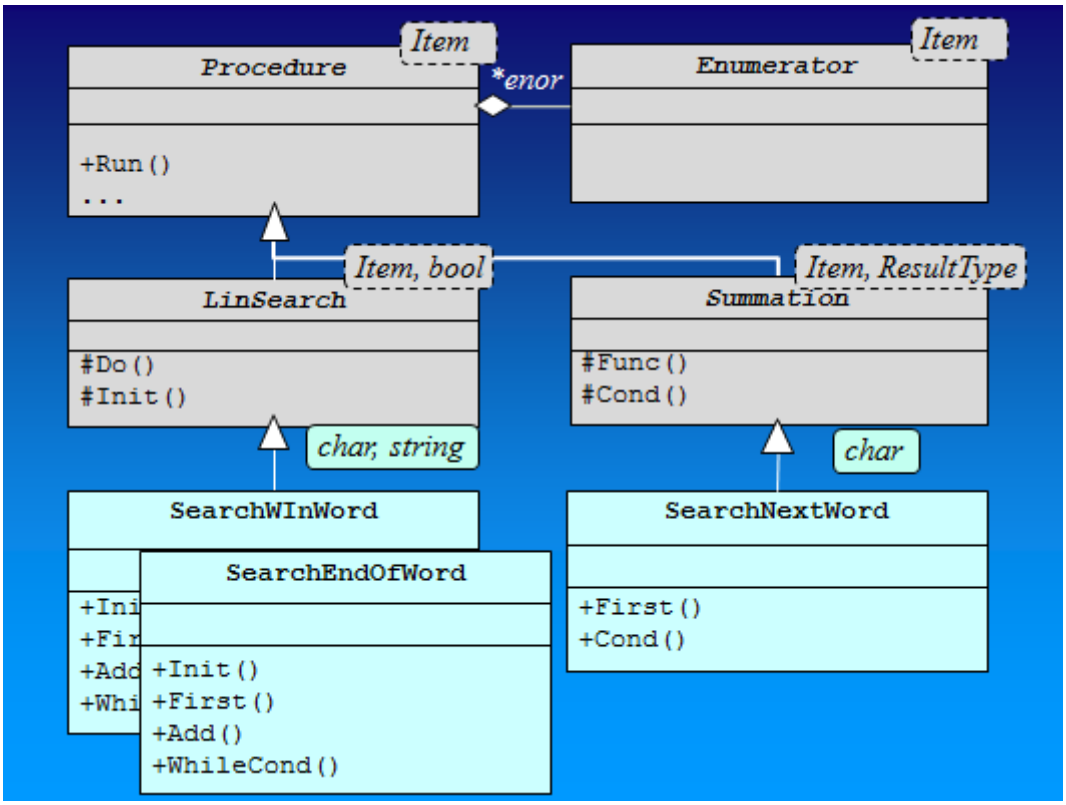
$$\textit{első\_szót\_keres(fit)} \quad \sim \quad l, \textit{fit} := \underset{i=1}{\overset{|\textit{fit}'|}{\textbf{search}}} \; \textit{fit}_i' \notin \textit{elválasztójel}$$

$$\textit{W\_keres(fit)} \quad \sim \quad \textit{akt.str}, \textit{fit} := \bigoplus_{i}^{|\textit{fit}'| \vee \textit{fit}_i' \in \{W\} \cup \textit{elválasztójel}} < \textit{fit}_i' > \qquad \textit{akt.vanW} := \textit{fit}_i = \text{'W'}$$

$$\textit{szóvégét\_keres(fit)} \quad \sim \quad \textit{akt.str}, \textit{fit} := \bigoplus_{i}^{|\textit{fit}'| \vee \textit{fit}_i' \in \textit{elválasztójel}} < \textit{fit}_i' >$$

$$\textit{következő\_szót\_keres(fit)} \quad \sim \quad l, \textit{fit} := \underset{i}{\overset{|\textit{fit}'|}{\textbf{search}}} \; \textit{fit}_i' \notin \textit{elválasztójel}$$

A felsoroló típusának implementálásához tehát szükség van a lineáris keresés és az összegzés programozási tételeinek osztály-sablonjaira (lásd előadás) is.



Megvalósítás

A megvalósításhoz felhasználjuk az előadáson bemutatott osztály-sablon könyvtárat. Azoknak fejállományait bemásoljuk a programunkba. Így csak a fenti osztálydiagramok kék színnel jelölt osztályait kell csak nekünk elkészíteni.[1]

MyMAxSearch (main.cpp)

Mindenekelőtt definiáljuk a jelölt szavak típusát. Erre ki kell terjeszteni a kiíró operátort is.

```
struct Word{
    string  str;
    bool    hasW;
    friend ostream& operator<<(ostream&, const Word&);
};

ostream& operator<<(ostream& f, const Word& df)
{
   f << df.str;
   return f;
}
```

A feladatot megoldó feltételes maximumkeresést (`MyMaxSearch`) az általános maximumkeresésből származtatjuk a `Word` segítségével történt példányosítás után.

```
class MyMaxSearch : public MaxSearch<Word>
{
   protected:
      int  Func(const Word& e) const{ return e.str.size();}
      bool Cond(const Word& e) const{ return e.hasW;}
};

int main()
{
   MyMaxSearch pr;
   WordEnumerator wit("text.txt");
   pr.AddEnumerator(&wit);

   pr.Run();

   if (pr.Found())
      cout << "A leghosszabb W-t tartalmazó szó "
           << pr.OptItem() << endl;
   else
      cout << "Nincs W-t tartalmazó szó\n";

   return 0;
}
```

[1] Az itt bemutatott kódnak nem is kellene a dokumentációban szerepelni, hiszen ez a forrásprogramban elolvasható.

Az absztrakt felsoroló osztályát is nekünk kell definiálni.

```
class WordEnumerator : public Enumerator<Word>
{
   protected:
      SeqInFileEnumerator<char> * fit;
      Word current;
      bool end;

   public:
      WordEnumerator(const std::string& str)
         {
            fit = new SeqInFileEnumerator<char>(str);
         }

      void First();
      void Next();
      bool End()     const { return end;}
      Word Current() const { return current; }

      WordEnumerator(){ delete fit;}
};
```

A `First()` és `Next()` metódusához szükség van a következő szó elejét megtaláló (`SearchNextWord`), az adott szóban 'W'-t kereső, de közben a szó betűit összegyűjtő (`SearchWInWord`) és a szó végét kereső, de közben a szó betűit összegyűjtő (`SerachEndOfWord`) tevékenységekre. Az első tevékenység osztályát a `LinSearch` osztály-sablonból, a másik kettőt a `Summation` osztály-sablonból származtatjuk.

A `Next()` metódus a tervezésnél leírt módon működik.

```
void WordEnumerator::Next()
{
   if(end = fit->End()) return;

   SearchWInWord pr1;
   pr1.AddEnumerator(fit);
   pr1.Run();
   current.hasW = fit->Current() == 'W';
   current.str = pr1.Answer();

   SearchEndOfWord pr2;
   pr2.AddEnumerator(fit);
   pr2.Run();
   current.str += pr2.Answer();

   SearchNextWord pr3;
   pr3.AddEnumerator(fit);
   pr3.Run();
}
```

A `First()` metódus a legelső szó megkeresése után a `Next()` metódussal azonos módon működik. A legelső szó elejét is a `SearchNextWord` tevékenységgel keressük meg,

de nem szabad megfeledkezni arról, hogy ez előtt el kell indítani a karakterenként felsorolót (`fit->First()`).Ezt mutatja az alábbi kód.

```
void WordEnumerator::First()
{
   fit->First();

   SearchNextWord pr;
   pr.AddEnumerator(fit);
   pr.Run();

   Next();
}
```

Végezetül definiáljuk a `SearchNextWord`, `SearchWInWord` és `SerachEndOfWord` tevékenységeket. Mindhárom osztályban felüldefiniáljuk a `First()` metódust, hiszen nem szabad a karakterenkénti felsorolást újrakezdeni.

```
class SearchNextWord : public LinSearch<char>
{
    protected:
        bool Cond(const char& e) const
           {
              return e!=' ' && e!='\t' && e!='\n';
           }
        void First(){}
};

class SearchWInWord : public Summation<char, string>
{
    protected:
        void Add(const char& e) { *res += e;}
        void Init(){ *res = "";}
        bool WhileCond(const char& e) const
           {
              return e!='W' && e!=' ' && e!='\t' && e!='\n';
           }
        void First(){}
};

class SearchEndOfWord : public Summation<char, string>
{
    protected:
        void Add(const char& e) { *res += e;}
        void Init(){ *res = "";}
        bool WhileCond(const char& e) const
           {
              return e!=' ' && e!='\t' && e!='\n';
           }
        void First(){}
};
```

Tesztelés

Fekete doboz tesztesetek:

1. Üres szöveg
2. Csak elválasztó jeleket tartalmazó szöveg.
3. Csupa ’W’ betűt nem tartalmazó szó.
4. Több szó, köztük egy ’W’ betűt tartalmazó szó a szöveg elején.
5. Több szó, közük egy ’W’ betűt tartalmazó szó a szöveg elején, előtte elválasztó jelek.
6. Több szó, közük egy ’W’ betűt tartalmazó szó a szöveg végén.
7. Több szó, közük egy ’W’ betűt tartalmazó szó a szöveg végén, utána elválasztó jelek.
8. Több szó, köztük azonos hosszúságú ’W’ betűt tartalmazó szavak.
9. Több szó, köztük azonos hosszúságú ’W’ betűt tartalmazó szavak, valamint egy ezeknél hosszabb ’W’ betűt tartalmazó szó.
10. Több szó, köztük azonos hosszúságú ’W’ betűt tartalmazó szavak, valamint egy ezeknél hosszabb ’W’ betűt tartalmazó szó a szöveg végén.
11. Több szó, köztük azonos hosszúságú ’W’ betűt tartalmazó szavak, valamint egy ezeknél hosszabb ’W’ betűt tartalmazó szó a szöveg elején.

Fehér doboz tesztesetek:

A saját kódrészben egyetlen elágazás található az eredmény kiírásánál (ennek mindkét ágát befutották a fekete doboz tesztesetek), ezen kívül csak szekvencia szerkezetet látunk, ezért nincs szükség külön fehér doboz tesztre.