

Trabajo práctico : Programación 1

Descripción del problema

“Dado un texto, que puede ser ingresado por teclado o por archivo, mostrar en pantalla una lista de las palabras con su frecuencia de ocurrencia. Requerimientos: Composición, Utilizar <sequence.hpp>”

Estructura del programa

En la siguiente figura, se presenta el esquema de composición para obtener la solución al problema propuesto. Cada bloque o estructura, está sustentada en mecanismos de delegación, por lo tanto, las funcionalidades son redireccionadas hacia el bloque padre, que se comporta como una caja negra. De esta manera, cada estructura maneja su propio nivel de abstracción.

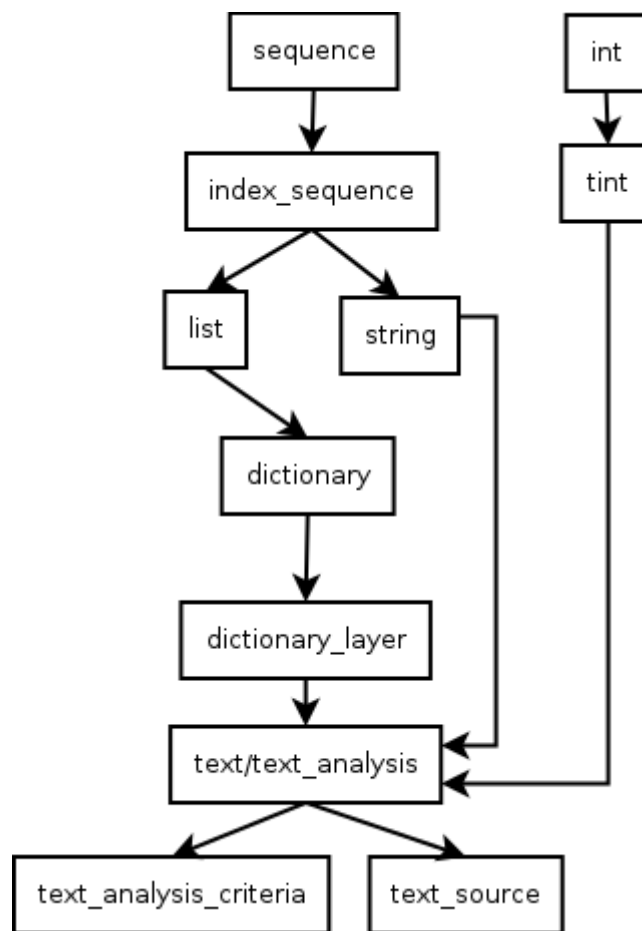


Figura 1: Esquema de composición

El programa fue escrito y compilado en C++, sin embargo es mayoritariamente estilo C, dado que no se hace uso de Clases, ni de las capacidades expresivas de C++. Esto significa, que las estructuras internas están construidas por <structs> que si bien ofrecen el mecanismo de abstracción de datos, no presentan ocultamiento de información. Es por ello, que en el código se indica con comentarios <public> y <private>, aquellas partes de las estructuras por las cuales éstas pueden ser accedidas. Sin embargo, hay que destacar que esto es solo una convención y no resulta ningún mecanismo seguro.

Para la resolución del problema, no solo basta construir las estructuras por composición, sino que también es necesario diseñar una política de requerimientos o protocolo, que se debe cumplir necesariamente para su funcionamiento. Cuando las estructuras son compuestas sin seguir ningún protocolo, todas las funcionalidades que se encuentren incluídas, que dependan de mecanismos de comparación, asignación, etc tienden por defecto a ser correctas cuando éstas trabajan con los tipos de dato primitivos (char, int, float..) ya que los operadores (== , = ..) se encuentran definidos para ellos.

Sin embargo, si uno construye un tipo de dato definido por usuario, como resulta en este caso (string) que consiste en una cadena de caracteres, estos operadores no pueden trabajar, porque no están definidos. Con lo cual, resulta un problema inherente a las estructuras elaboradas. Para solucionar este asunto, o bien se plantea re-escribir todas las composiciones encapsulando todos los operadores, sacrificando que puedan funcionar con tipos de dato primitivo, o re-adaptar sólo la estructura que se quiera utilizar para que trabaje con un tipo definido por usuario. En ambos casos, adhiriendo a una misma política uniforme de requerimientos de diseño.

Por una cuestión de minimizar el costo de tiempo momentáneo, se planteó solamente re-adaptar la estructura necesaria para la resolución del problema.

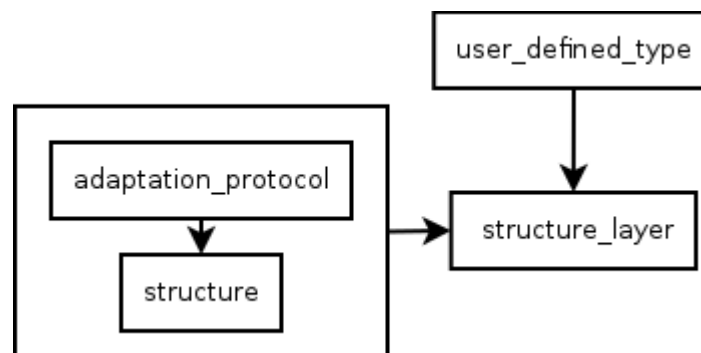


Figura 2: Proceso de re-adaptación de estructura para trabajar con tipos definidos por usuario.

La política que debe adherir tanto el tipo definido por usuario, como la estructura con la que debe trabajar, es la siguiente:

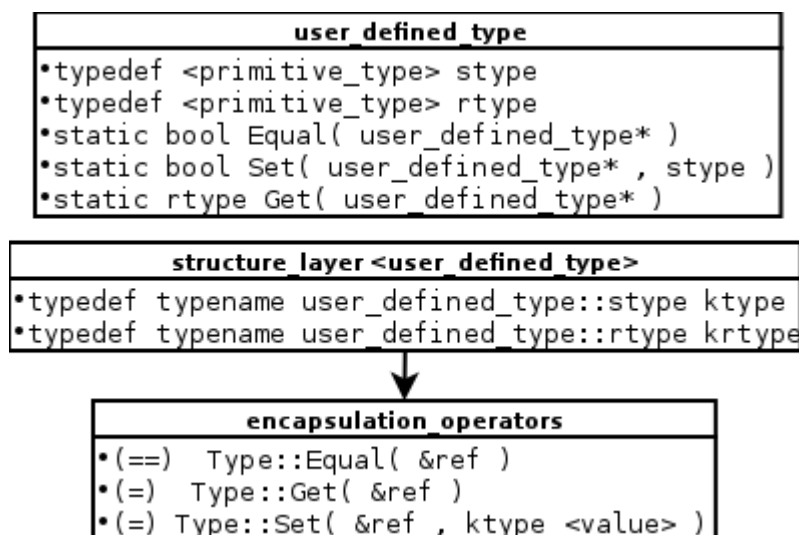


Figura 3: Requerimiento / protocolo

El tipo definido por usuario, debe ser capaz de mostrar un nombre único `<stype>` que resulta un sinónimo de un tipo primitivo, y no debe cambiar. Por otro lado, debe mostrar los nombres que encapsulan los operadores de comparación, y asignación: `Equal()`, `Get()`, `Set()`.

Para el proceso de adaptación: Desde la estructura contenedora de datos, que se encuentra parametrizada, las líneas `<typedef>` de requerimiento, tienen el objetivo de tomar el nombre del tipo definido por usuario y bajo el operador de contexto obtener el nombre de la representación primitiva y asignar el sinónimo `<ktype>`. Posteriormente, se encapsulan todos los operadores.

Lo que resulta de esta construcción, es que la estructura del tipo definido por usuario es siempre la que se encarga de efectuar las operaciones dependientes del tipo, y no la estructura contenedora de datos. Ya que la misma “delega” esta función.

**En el programa, sólo se aplicó este proceso de adaptación para el diccionario `<dictionary-layer.hpp>`*

Funcionamiento del programa

El programa permite analizar un texto que puede ser ingresado desde una fuente determinada. El método de uso, es muy sencillo y consiste en proveer como parámetro: La fuente de texto, y el método de análisis.

En este caso, por las condiciones del problema, solamente se implementaron las fuentes de texto: `<FromFile>` y `<FromKeyboard>`, al igual que un único método de análisis `<WordsFrequency>` (Opcionalmente, está permitida la opción `<None>` que no aplica ningún análisis al texto cargado. El código permite incluir más fuentes proveedoras de texto en: `<TextSource.hpp>` y métodos de análisis en `<AnalysisCriteria.hpp>`.

```
//main.cpp

#include "text.hpp"

using namespace std;

int main(int argc, char* argv[])
{
    Text book; book.Init();
    book.GetText<FromKeyboard, WordsFrequency>();
    book.PrintResult();

    return 0;
}
```

Figura 4: Vista del fuente 'main.cpp' que resuelve el problema dado.

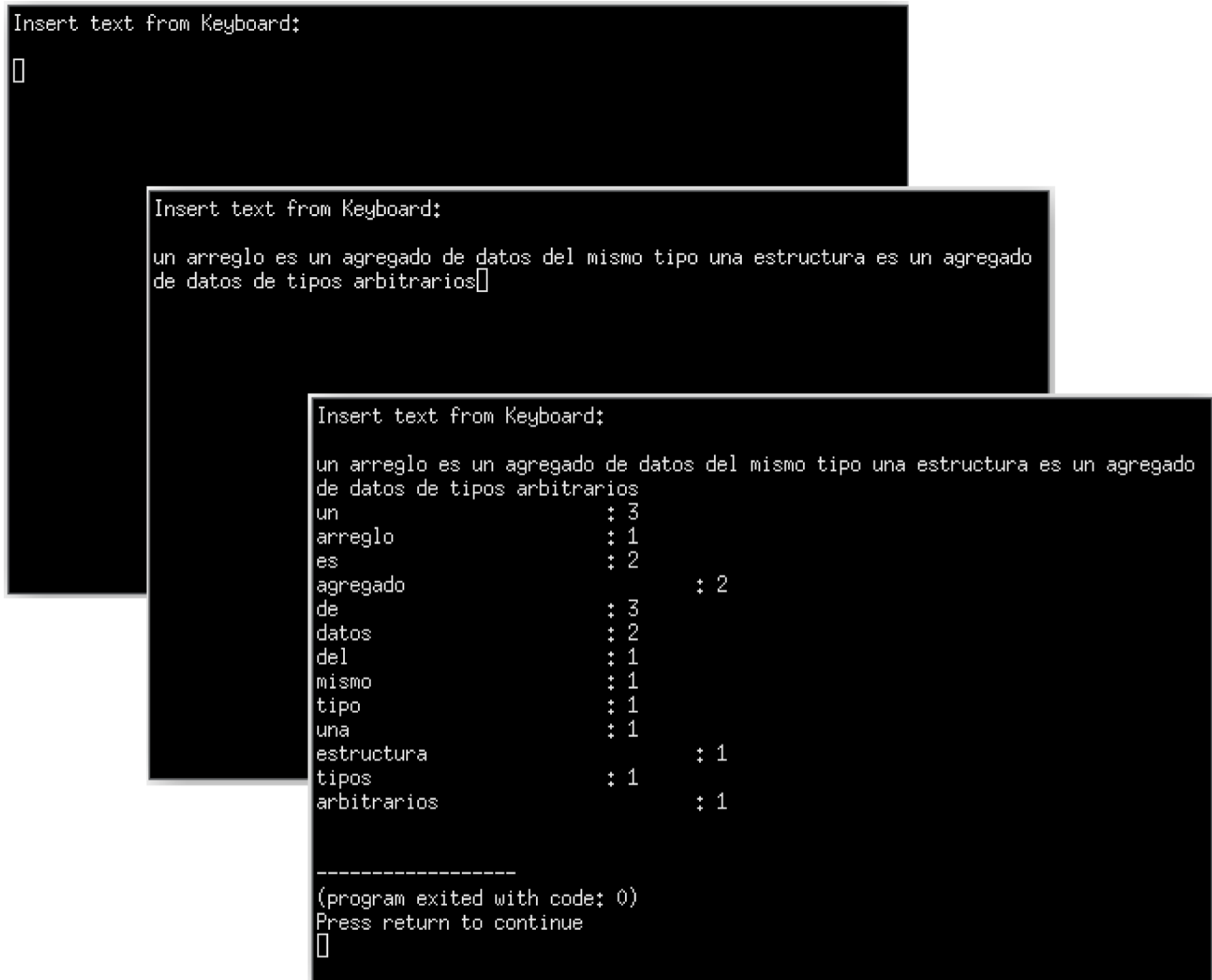
Testeo del programa I

A continuación se presentan screenshots que comprueban el funcionamiento del programa:

Caso 1: `book.GetText<FromKeyboard, WordsFrequency>()`;

Input:

“un arreglo es un agregado de datos del mismo tipo una estructura es un agregado de tipos arbitrarios”



```
Insert text from Keyboard:
[]

Insert text from Keyboard:
un arreglo es un agregado de datos del mismo tipo una estructura es un agregado
de datos de tipos arbitrarios[]

Insert text from Keyboard:
un arreglo es un agregado de datos del mismo tipo una estructura es un agregado
de datos de tipos arbitrarios
un          : 3
arreglo     : 1
es          : 2
agregado    : 2
de          : 3
datos      : 2
del         : 1
mismo      : 1
tipo       : 1
una        : 1
estructura : 1
tipos      : 1
arbitrarios: 1

-----
(program exited with code: 0)
Press return to continue
[]
```

Figura 5: Test de funcionamiento para entrada de texto por teclado.

Testeo del programa II

Caso 2: `book.GetText<FromFile, WordsFrequency>()`;

Input: `<input.txt>`

"this is a text example to test the application written in c++, it shows (if selected) the frequency of any word. it use a space as a delimiter and there is no exclusion of special characters the new line is the END of this text."

```
Loading text from file <input.txt>

this           : 2
is             : 3
a              : 3
text           : 1
example        : 1
to             : 1
test           : 1
the            : 4
application    : 1
written        : 1
in             : 1
c++,           : 1
it             : 2
shows          : 1
(if            : 1
selected)      : 1
frequency      : 1
of             : 3
any            : 1
word.          : 1
use            : 1
space          : 1
as             : 1
delimiter      : 1
and            : 1
there          : 1
no             : 1
exclusion       : 1
special        : 1
characters     : 1
new            : 1
line           : 1
END            : 1
text.          : 1

-----
(program exited with code: 0)
Press return to continue
█
```

Figura 6: Test de funcionamiento para entrada de texto por archivo.

Observaciones

Si bien el programa, resuelve el problema dado. Posee limitaciones que son propias a la implementación (mejorables), y otras que son inherentes al código. Entre las limitaciones de implementación, cabe destacar, que el programa no es capaz de excluir símbolos de puntuación y gramaticales en la identificación de palabras, ya que utiliza únicamente como criterio la separación por espacios ' ' como delimitador. Por otro lado, el texto no puede ser tan extenso como se desee, ya que tiene fijado estáticamente un límite de caracteres. La entrada de texto no debe finalizar por espacio, ya que no se encuentran considerados los casos excepcionales. Estos detalles son mitigables sin problemas.

Existen otras limitaciones más profundas, entre ellas, una que resulta importante es que el programa no ofrece escalabilidad, y tiende a ser lento en tiempo de ejecución conforme aumente el límite de almacenamiento impuesto. Ésto se debe a que, la secuencia no es de acceso aleatorio, sino que es de acceso secuencial y el tiempo de acceso se incrementa linealmente con la longitud de la misma. Las mejoras posibles desde este lugar, es la redefinición de la estructura de datos, pero la propagación de los cambios pueden impactar notablemente en forma descendente.

Dentro de los detalles internos del código, se detectan operaciones que tienen un costo muy alto en recursos computacionales para efectuarse, que en caso de escalabilidad tenderían a ser varias veces más lentas. Una de ellas es la eliminación e inserción de elementos de Diccionarios, y Listas. Éstas consisten en operaciones de movimientos en bloque de elementos y no se recomienda utilizarlas.

Finalmente, desde el punto de vista del desarrollo de software, quizás las conclusiones y observaciones sean más importantes aún. El código como se ha mencionado anteriormente, resuelve el problema. Sin embargo, resulta inviable de cara al futuro por el hecho de que muestra varios puntos de alto mantenimiento, como por ejemplo, el proceso de adaptación para trabajar con otros tipos de datos (*Notar, que el tipo primitivo <int> se encuentra encapsulado para su uso <int.hpp>*), y además, existe una política/protocolo que deben satisfacer las partes para la cohesión del código, tanto internamente, como externamente en caso de vinculación con código de terceros. Esto es así, porque la fijación de protocolos conduce indirectamente a procesamiento de documentación, información, y en el mejor de los casos, se debería esperar que se cumpla en su totalidad. Todo esto, incrementa los costos de mantenimiento y la productividad se vería afectada considerablemente.

```
1  /*
2  * Matías Gastón Santiago
3  * TP Programación 1 -> Prof: Gabriel Pimentel
4  *
5  * source code :: main.cpp
6  *
7  */
8
9  #include "text.hpp"
10
11  using namespace std;
12
13  int main(int argc, char* argv[])
14  {
15
16      Text book; book.Init();
17
18      book.GetText<FromKeyboard, WordsFrequency>();
19
20      book.PrintResult();
21
22      return 0;
23  }
24
```

```
1  /*
2  * Matías Gastón Santiago
3  * TP Programación 1 -> Prof: Gabriel Pimentel
4  *
5  * source code :: text.hpp
6  *
7  */
8
9  #ifndef TEXT H
10 #define TEXT H
11
12 #include "TextAnalysis.hpp"
13 #include "TextSource.hpp"
14 #include "AnalysisCriteria.hpp"
15
16 #endif
17
```



```

1  /*
2  * Matías Gastón Santiago
3  * TP Programación 1 -> Prof: Gabriel Pimentel
4  *
5  * source code :: TextAnalysis.hpp
6  *
7  */
8
9  #ifndef TEXT ANALYSIS H
10 #define TEXT ANALYSIS H
11
12 #include <stdio.h>
13 #include "dictionary-layer.hpp"
14 #include "string.hpp"
15 #include "int.hpp"
16
17
18 struct Text
19 {
20     Dictionary<String, TInt> rep;
21     List<char> raw;
22     String word;
23     void Process(void);
24     bool Init(void);
25     void PrintResult(void);
26     void PrintText(void);
27
28     //Public
29     template <class Source, class Criteria> bool GetText(void)
30     {
31         Init();
32         Source::GetText( this );
33         Criteria::AnalysisMethod( this );
34         return true;
35     };
36
37 };
38
39 void Text::PrintText(void)
40 {
41     for ( int i=raw.Begin(); i<=raw.End(); i++ )
42     {
43         cout << *raw.Item(i);
44     }
45 }
46
47 void Text::PrintResult(void)
48 {
49     rep.Print();
50 }
51
52
53 bool Text::Init(void)
54 {
55     rep.Init();
56     raw.Init();
57     word.Init();
58
59     return true;
60 }
61
62
63
64 #endif
65
66

```

```

1  /*
2  * Matías Gastón Santiago
3  * TP Programación 1 -> Prof: Gabriel Pimentel
4  *
5  * source code :: TextSource.hpp
6  *
7  */
8
9  #ifndef TEXT_SOURCE_H
10 #define TEXT_SOURCE_H
11
12 #include <stdio.h>
13 #include "list.hpp"
14 #include "TextAnalysis.hpp"
15
16 struct FromKeyboard
17 {
18     static List<char> GetText( Text* save );
19 };
20
21 struct FromFile
22 {
23     static List<char> GetText( Text* save );
24 };
25
26 // ... add sources ... //
27
28 List<char> FromFile::GetText( Text* save )
29 {
30     save->raw.Init();
31
32     FILE* ptxt;
33     ptxt=fopen ( "input.txt", "r" );
34
35     char temp[1024];
36     int i=0;
37
38     if (ptxt==NULL) cout << "Error opening file" << endl;
39     else
40     {
41         cout << "Loading text from file <input.txt>" << endl << endl;
42         while((temp[i] = getc(ptxt))!='\n') i++ ;
43     }
44
45     int size=i;
46     for ( i=0; i<size; i++) save->raw.Add( temp[i] );
47
48     return save->raw;
49 }
50
51
52 List<char> FromKeyboard::GetText( Text* save )
53 {
54     save->raw.Init();
55
56     cout << "Insert text from Keyboard:" << endl << endl;
57     char temp[1024];
58
59     int i=0;
60     while((temp[i] = getchar())!='\n') i++ ;
61     int size=i;
62
63     for ( i=0; i<size; i++) save->raw.Add( temp[i] );
64
65     return save->raw;
66 }
67
68 #endif
69

```

```

1  /*
2  * Matías Gastón Santiago
3  * TP Programación 1 -> Prof: Gabriel Pimentel
4  *
5  * source code :: AnalysisCriteria.hpp
6  *
7  */
8
9  #ifndef ANALYSIS CRITERIA H
10 #define ANALYSIS CRITERIA H
11
12 #include <stdio.h>
13 #include "list.hpp"
14 #include "TextAnalysis.hpp"
15
16 struct WordsFrequency
17 {
18     static void AnalysisMethod( Text* text );
19 };
20
21 struct None
22 {
23     static void AnalysisMethod( Text* text );
24 };
25
26 // ... add analysis method ... //
27
28 void None::AnalysisMethod( Text* text )
29 {
30
31 }
32
33
34 void WordsFrequency::AnalysisMethod( Text* text )
35 {
36     int space count=0;
37     TInt var;
38     var.SetValue(1);
39     bool sucess;
40
41     for( int i=text->raw.Begin(); i<=text->raw.End(); i++)
42     {
43
44         if ( *(text->raw.Item(i)) != ' ' )
45         {
46             space count++;
47             text->word.ConcatChar( *(text->raw.Item(i)) );
48         }
49
50         else
51         {
52             sucess=text->rep.Add( text->word.GetValue(), var.GetValue());
53
54             if ( !sucess ) text->rep.Key( text->word.GetValue() )->Increment();
55             text->word.Init();
56         }
57     }
58
59     if ( space count==0 ) text->rep.Add( text->word.GetValue(), var.GetValue() );
60     sucess=text->rep.Add( text->word.GetValue(), var.GetValue() );
61
62     if ( !sucess ) text->rep.Key( text->word.GetValue() )->Increment();
63
64 }
65
66 #endif
67
68

```

```

1  /*
2  * Matías Gastón Santiago
3  * TP Programación 1 -> Prof: Gabriel Pimentel
4  *
5  * source code :: dictionary-layer.hpp
6  *
7  */
8
9  #ifndef DIC H
10 #define DIC H
11
12 #include "list.hpp"
13
14 template < class KType, class VType >
15 struct Dictionary
16 {
17     //Private
18     List<KType> repk;
19     List<VType> repv;
20     int d items;
21     bool initialized;
22
23     //Required to link KType/VType representation
24     typedef typename KType::stype ktype;
25     typedef typename VType::stype vtype;
26
27     //Public
28     bool Init( void ) ;
29     VType* Key(ktype key );
30
31     int Count(ktype key);
32     bool Add(ktype key, vtype value);
33     void Print(void);
34     int Size(void);
35     int Begin(void);
36     int End(void);
37 };
38
39 template < class KType, class VType >
40 int Dictionary<KType,VType>::Size(void)
41 {
42     return d items;
43 }
44
45 template < class KType, class VType >
46 int Dictionary<KType,VType>::Begin(void)
47 {
48     return 1;
49 }
50
51 template < class KType, class VType >
52 int Dictionary<KType,VType>::End(void)
53 {
54     return d items;
55 }
56
57 template < class KType, class VType >
58 void Dictionary<KType,VType>::Print(void)
59 {
60     if ( initialized )
61     {
62         for (int i=Begin(); i<=End(); i++)
63         {
64             cout << KType::Get(repk.Item(i)) << "\t\t\t : " << VType::Get(repv.Item(i)) << "\n";
65             endl;
66         }
67     }
68 }

```

```

69 template <class KType, class VType >
70 VType* Dictionary<KType, VType>::Key(ktype key)
71 {
72     KType tkey;
73
74     int ref;
75     for (int i=repk.Begin(); i<=repk.End(); i++)
76     {
77         KType::Set(&tkey, key);
78         if ( KType::Equal( &tkey, repk.Item(i) ) ) ref=i;
79     }
80
81     return repv.Item(ref);
82 }
83
84
85
86 template < class KType, class VType >
87 bool Dictionary<KType,VType>::Add(ktype key, vtype value)
88 {
89     bool sucess=false;
90     int counter=0;
91
92     KType tkey;
93     VType tvalue;
94     KType::Set( &tkey, key );
95     VType::Set( &tvalue, value );
96
97     for (int i=repk.Begin(); i<=repk.End(); i++)
98     {
99         KType::Set( &tkey, key );
100         if ( KType::Equal( repk.Item(i), &tkey ) ) counter++;
101     }
102
103     if ( counter == 0 )
104     {
105         repk.Add( tkey );
106         repv.Add( tvalue );
107         d items++;
108         sucess=true;
109     }
110     else
111     {
112         sucess=false;
113     }
114
115     return sucess;
116 }
117
118
119
120 template < class KType, class VType >
121 int Dictionary<KType, VType>::Count(ktype key)
122 {
123     int counter=0;
124
125     KType tkey;
126     KType::Set( &tkey, key );
127
128
129     for (int i=repk.Begin(); i<=repk.End(); i++)
130     {
131         KType::Set( &tkey, key );
132         if ( KType::Equal( repk.Item(i), &tkey ) ) counter++;
133     }
134
135     return counter;
136 }
137

```

```
138  template < class KType, class VType >
139  bool Dictionary<KType,VType>::Init(void)
140  {
141      bool sucess;
142      d items=0;
143      sucess=repk.Init();
144      initialized=true;
145      return (sucess)? repv.Init() : false;
146  }
147
148  #endif
149
```

```

1  /*
2  * Matías Gastón Santiago
3  * TP Programación 1 -> Prof: Gabriel Pimentel
4  *
5  * source code :: dictionary.hpp
6  *
7  */
8
9  #ifndef DIC H
10 #define DIC H
11
12 #include "list.hpp"
13
14 template < class KType, class VType >
15 struct Dictionary
16 {
17     //Private
18     List<KType> repk;
19     List<VType> repv;
20     int d items;
21     bool initialized;
22     bool addpair(KType key, VType value);
23
24     //Public
25     bool Init( void ) ;
26     VType* Key(KType key );
27
28
29     bool Add(KType key, VType value);
30     bool Check(KType key);
31     void Print(void);
32     void PrintDebug(void);
33     int Size(void);
34     int Begin(void);
35     int End(void);
36 };
37
38
39 template < class KType, class VType >
40 void Dictionary<KType,VType>::PrintDebug(void)
41 {
42     if ( initialized )
43     {
44         for (int i=Begin(); i<=End(); i++)
45         {
46             cout << repk.Item(i) << " :: " << *repk.Item(i) << "\t-> " << repv.Item(i) <<
47                 " :: " << *repv.Item(i) << endl;
48         }
49     }
50
51 template < class KType, class VType >
52 int Dictionary<KType,VType>::Size(void)
53 {
54     return d items;
55 }
56
57 template < class KType, class VType >
58 int Dictionary<KType,VType>::Begin(void)
59 {
60     return 1;
61 }
62
63 template < class KType, class VType >
64 int Dictionary<KType,VType>::End(void)
65 {
66     return d items;
67 }
68

```

```

69 template < class KType, class VType >
70 void Dictionary<KType,VType>::Print(void)
71 {
72     if ( initialized )
73     {
74         for (int i=Begin(); i<=End(); i++)
75         {
76             cout << *repk.Item(i) << "\t : " << *repv.Item(i) << endl;
77         }
78     }
79 }
80
81
82 template <class KType, class VType >
83 VType* Dictionary<KType, VType>::Key(KType key)
84 {
85     int ref=0;
86
87     for ( int i=repk.Begin(); i<=repk.End(); i++ )
88     {
89         if ( *repk.Item(i) == key ) ref=i;
90     }
91
92     return repv.Item( ref );
93 }
94
95
96 template < class KType, class VType >
97 bool Dictionary<KType,VType>::Check(KType key)
98 {
99     return repk.Check(key);
100 }
101
102 template < class KType, class VType >
103 bool Dictionary<KType,VType>::Add(KType key, VType value)
104 {
105     bool sucess=false;
106
107     sucess=addpair(key, value);
108     d items++;
109
110     return sucess;
111 }
112
113 template < class KType, class VType >
114 bool Dictionary<KType,VType>::Init(void)
115 {
116     bool sucess;
117     d items=0;
118     sucess=repk.Init();
119     initialized=true;
120     return (sucess)? repv.Init() : false;
121 }
122
123 template < class KType, class VType >
124 bool Dictionary<KType,VType>::addpair( KType key, VType value)
125 {
126     bool sucess=false;
127     sucess=repk.Add(key);
128     return ( sucess )? repv.Add(value) : false;
129 }
130 #endif
131

```



```

1  /*
2  * Matías Gastón Santiago
3  * TP Programación 1 -> Prof: Gabriel Pimentel
4  *
5  * source code :: list.hpp
6  *
7  */
8
9  #ifndef LIST H
10 #define LIST H
11
12 #include "idx sequence.hpp"
13
14 template < class Type >
15 struct List
16 {
17     //Private
18     IdxSequence<Type> rep;
19     int list items;
20     bool ValidItem(int pos);
21
22     //Public
23     bool Init( void ) ;
24     Type* Item(int pos);
25
26
27     bool Add(Type value);
28     bool CreateWith( List<Type> list );
29     bool Insert(int pos, Type value );
30     bool Remove(int pos);
31
32     void Swap(int pos s, int pos f);
33
34
35     int CountCheck(Type value);
36
37     int Begin(void);
38     int End(void);
39     int Size( void );
40
41     void Print(void);
42 };
43
44 template <class Type>
45 bool List<Type>::CreateWith( List<Type> list )
46 {
47     bool sucess=false;
48
49     for (int i=list.Begin(); i<=list.End(); i++)
50     {
51         sucess=Add( *list.Item(i) );
52     };
53
54     return sucess;
55 }
56
57 template <class Type>
58 bool List<Type>::Add(Type value)
59 {
60     list items++;
61     return rep.Add( value );
62 };
63
64 template <class Type>
65 void List<Type>::Swap(int pos s, int pos f)
66 {
67     rep.Move( pos s, pos f);
68 }
69

```

```

70  template <class Type>
71  bool List<Type>::Insert( int pos, Type value )
72  {
73      bool sucess=ValidItem(pos);
74
75      if ( sucess )
76      {
77          List A; A.Init();
78          for ( int i=A.Begin(); i<pos;i++) A.Add( Item(i) );
79          A.Add( value );
80          for ( int i=pos; i<=Size();i++) A.Add( Item(i) );
81          Init();
82          for ( int i=Begin(); (i<=A.End()) && (sucess) ;i++) sucess=Add( A.Item(i) );
83          return sucess;
84      }
85      else return !Add(value);
86  }
87
88
89  template <class Type>
90  int List<Type>::Begin(void)
91  {
92      return 1;
93  }
94
95  template <class Type>
96  int List<Type>::End(void)
97  {
98      return list items;
99  }
100
101  template <class Type>
102  bool List<Type>::Init( void )
103  {
104      list items=0;
105      return rep.Init();
106  }
107
108  template < class Type >
109  int List<Type>::Size(void)
110  {
111      return rep.Size();
112  }
113
114  template < class Type >
115  bool List<Type>:: ValidItem(int pos)
116  {
117      return ( (pos>=Begin()) && (pos<=End()) )? true : false;
118  };
119
120
121  template <class Type>
122  bool List<Type>::Remove( int pos )
123  {
124      if ( ValidItem(pos) )
125      {
126          List A; A.Init();
127          for ( int i=A.Begin(); i<pos;i++) A.Add( *Item(i) );
128          for ( int i=(pos+1); i<=Size();i++) A.Add( *Item(i) );
129          Init();
130          for ( int i=Begin(); i<=A.Size();i++) Add( *A.Item(i) );
131          return true;
132      }
133      else
134      {
135          return false;
136      }
137  }
138

```

```
139 template <class Type>
140 Type* List<Type>::Item(int pos)
141 {
142     return rep.Item( pos );
143 }
144
145
146
147
148 template <class Type>
149 int List<Type>::CountCheck(Type value)
150 {
151     int counter=0;
152
153     for ( int i=Begin(); i<=End() ; i++)
154     {
155         if ( value == *Item(i) ) counter++;
156     }
157
158     return counter;
159
160     /*
161     * return 0 -> No item on List
162     * return {1..n} -> Exists n coincidences
163     */
164 }
165
166
167 template <class Type>
168 void List<Type>::Print(void)
169 {
170     for (int i = Begin(); i <= End(); i++ ) cout << *Item(i) << endl;
171 }
172
173 #endif
174
```

```

1  /*
2  * Matías Gastón Santiago
3  * TP Programación 1 -> Prof: Gabriel Pimentel
4  *
5  * source code :: string.hpp
6  *
7  */
8
9  #ifndef STRINGS H
10 #define STRINGS H
11
12 #include "idx sequence.hpp"
13
14 struct String
15 {
16     //Private
17     IdxSequence<char> rep;
18     bool AddChar(char value);
19     char GetChar(int pos);
20     int Lenght( char const* value );
21     bool Init( void );
22
23     //Required to link
24     typedef char const* rtype;
25     typedef char const* stype;
26
27     //Private
28     stype GetValue(void);
29     bool SetValue( stype value );
30
31     //Public
32     static bool Equal( String* pstring a, String* pstring b );
33     static bool Set( String* pstring, stype value );
34     static rtype Get( String* pstring );
35     int Begin(void);
36     int End(void);
37     int Size( void );
38     void Clear( void );
39     void SwapChar( int pos s, int pos f);
40     void RplChar(int pos s, char value);
41     bool Concat( stype value );
42     bool ConcatChar( char value );
43     bool ConcatStp( int pos s, stype value, int pos f );
44     stype GetSel( int pos s, int pos f);
45     bool Insert( stype value, int pos );
46 };
47
48 bool String::Set( String* pstring, stype value )
49 {
50     return pstring->SetValue( value );
51 }
52
53 typename String::rtype String::Get( String* pstring )
54 {
55     return pstring->GetValue();
56 }
57
58 bool String::Equal( String* A, String* B)
59 {
60     int count=0;
61
62     if ( A->Size()==B->Size() )
63     {
64         for (int i=1; (i<=A->Size()); i++) if ( A->GetChar(i)==B->GetChar(i) ) count++;
65     }
66
67     return ( count == A->Size() )? true : false;
68 }
69

```

```

70 bool String::AddChar(char value)
71 {
72     return rep.Add( value );
73 }
74
75 char String::GetChar(int pos)
76 {
77     return *rep.Item(pos);
78 }
79
80 bool String::Init( void )
81 {
82     return rep.Init();
83 }
84
85 int String::Lenght( stype value )
86 {
87     int nchars;
88     for (nchars=0; *value != '\0';value++) nchars++;
89     return nchars;
90 }
91
92 String::stype String::GetValue(void)
93 {
94     return rep.First();
95 };
96
97 bool String::SetValue( stype value )
98 {
99     bool sucess = rep.Init();
100     for (int i=0; (i<Lenght(value)) && (sucess); i++)
101     {
102         sucess = AddChar( value [i] );
103     }
104     return sucess;
105 };
106
107 int String::Size( void )
108 {
109     return rep.Size();
110 }
111
112 int String::Begin(void)
113 {
114     return 1;
115 }
116
117 int String::End(void)
118 {
119     return Size();
120 }
121
122 String::stype String::GetSel( int pos s, int pos f)
123 {
124     String A;
125     A.Init();
126
127     for (int i=pos s; i<=pos f; i++) A.AddChar( GetChar(i) );
128
129     return A.GetValue();
130 }
131
132 bool String::Concat( stype value )
133 {
134     bool sucess;
135     for (int i=0; i<Lenght(value); i++) sucess=AddChar( value[i] );
136     return sucess;
137 }
138

```

```

139 bool String::ConcatChar( char value )
140 {
141     bool sucess;
142     sucess=AddChar( value );
143     return sucess;
144 }
145
146 bool String::ConcatStp(int pos s, stype value , int pos f)
147 {
148     bool sucess;
149     for (int i=pos s-1; i<pos f; i++) sucess=AddChar( value[i] );
150     return sucess;
151 }
152
153 bool String::Insert( stype value, int pos )
154 {
155     String result;
156     result.Init();
157
158     result.ConcatStp( Begin(), GetValue(), pos-1 );
159     result.Concat( value );
160     result.ConcatStp( pos, GetValue(), End() );
161
162     return SetValue( result.GetValue() );
163 }
164
165 void String::SwapChar( int pos1, int posf )
166 {
167     rep.Move(pos1, posf);
168 }
169
170 void String::RplChar( int pos s, char value )
171 {
172     *rep.Item(pos s) = value;
173 }
174
175 void String::Clear( void )
176 {
177     rep.Init();
178 }
179
180 #endif
181

```

```

1  /*
2  * Matías Gastón Santiago
3  * TP Programación 1 -> Prof: Gabriel Pimentel
4  *
5  * source code :: idx sequence.hpp
6  *
7  */
8
9  #ifndef IDX SEQUENCE H
10 #define IDX SEQUENCE H
11
12 #include "sequence.hpp"
13
14 template < class Type >
15 struct IdxSequence
16 {
17     //Private
18     Sequence<Type> rep;
19     bool ValidPosition( int pos );
20
21     //Public
22     bool Init( void ) ;
23     Type* Item(int pos);
24     bool Add(const Type& item ) ;
25     void Move(int source pos, int target pos);
26
27     Type* First( void );
28     int Size(void);
29
30 };
31
32 template <class Type>
33 bool IdxSequence<Type>::ValidPosition( int pos )
34 {
35     bool result;
36     if ( (pos >= 1) && (pos <= Size()) ) result=true;
37     else result=false;
38
39     return result;
40 }
41
42 template <class Type>
43 bool IdxSequence<Type>::Init( void )
44 {
45     return rep.Initialize();
46 }
47
48 template <class Type>
49 Type* IdxSequence<Type>::First( void )
50 {
51     return rep.First();
52 }
53
54 template <class Type>
55 Type* IdxSequence<Type>::Item( int pos )
56 {
57     Type* result=rep.First();
58     for ( int i=1; i<pos; i++ ) result=rep.Next();
59     return result;
60 }
61
62
63
64
65
66
67
68
69

```

```
70 template <class Type>
71 void IdxSequence<Type>::Move( int source pos, int target pos )
72 {
73     Type source value=*Item(source pos);
74     Type target value=*Item(target pos);
75
76     for( int i=source pos; i<target pos; i++) *Item(i-1) = *Item(i);
77     *Item(target pos)=source value;
78     *Item(target pos-1)=target value;
79 }
80
81 template <class Type>
82 bool IdxSequence<Type>::Add(const Type& item)
83 {
84     return rep.Add( item );
85 }
86
87
88 template <class Type>
89 int IdxSequence<Type>::Size(void)
90 {
91     return rep.Count();
92 }
93
94
95
96
97 #endif
98
```



```

1  /*
2  * Matías Gastón Santiago
3  * TP Programación 1 -> Prof: Gabriel Pimentel
4  *
5  * source code :: sequence.hpp
6  *
7  */
8
9  #ifndef SEQUENCE H
10 #define SEQUENCE H
11
12 #include <iostream>
13 #include <memory.h>
14
15
16 using namespace std;
17
18 template < class Type >
19 struct Sequence
20 {
21     //Private
22     typedef bool (*Predicate)(Type item);
23     enum { SEQUENCE MAX ITEMS = 512, SEQUENCE EMPTY = -1 } ;
24     Type rep[ SEQUENCE MAX ITEMS ] ;
25     int first ;
26     int last ;
27     Type* current ;
28     bool initialized ;
29
30     //Public
31     bool Initialize( void ) ;
32     bool CreateWith( Type const data[], int size ) ; //modif agregado "const"
33     int Count( void ) ;
34     void Select( Predicate predicate, Sequence& output ) ;
35     bool IsEmpty( void ) ;
36     bool IsFull( void ) ;
37     bool Add( const Type& item ) ;
38     Type* First( void ) ;
39     Type* Next( void ) ;
40     void Dump( void ) ;
41     void Free( void ) ;
42
43 } ;
44
45
46 template< class Type >
47 bool Sequence<Type>::Initialize( void )
48 {
49     ::memset( rep, 0, SEQUENCE MAX ITEMS ) ;
50     first = last = SEQUENCE EMPTY ;
51     current = 0 ;
52     initialized = true ;
53
54     return initialized ;
55 }
56
57 template< class Type >
58 bool Sequence<Type>::CreateWith( Type const data[], int size )
59 {
60     bool success = Initialize() ;
61     for( int index = 0; ( index < size ) && ( success ); index++ )
62     {
63         success = Add( data[ index ] ) ;
64     }
65     return success ;
66 }
67
68
69

```

```

70 template< class Type >
71 int Sequence<Type>::Count( void )
72 {
73     return initialized ? last + 1 : 0 ;
74 }
75
76 template< class Type >
77 bool Sequence<Type>::IsEmpty( void )
78 {
79     return initialized ? ( first == SEQUENCE_EMPTY ) : true ;
80 }
81
82 template< class Type >
83 bool Sequence<Type>::IsFull( void )
84 {
85     return initialized ? ( last == SEQUENCE_MAX_ITEMS ) : true ;
86 }
87
88 template< class Type >
89 bool Sequence<Type>::Add( const Type& item )
90 {
91     bool result = false ;
92     if ( initialized ) {
93         if ( !IsFull() ) {
94             rep[ ++last ] = item ;
95             if ( IsEmpty() ) {
96                 first = last ;
97             }
98             result = true ;
99         }
100     }
101     return result ;
102 }
103
104 template< class Type >
105 Type* Sequence<Type>::First( void )
106 {
107     return ( !IsEmpty() ) ? ( current = rep ) : 0 ;
108 }
109
110 template< class Type >
111 Type* Sequence<Type>::Next( void )
112 {
113     Type* result = 0 ;
114     if ( initialized ) {
115         current++ ;
116         if ( current <= &rep[ last ] ) result = current ;
117     }
118     return result ;
119 }
120
121 template< class Type >
122 void Sequence<Type>::Dump( void )
123 {
124     Type* item = First() ;
125     while ( item ) {
126         cout << *item << " " ;
127         item = Next() ;
128     }
129 }
130
131 template< class Type >
132 void Sequence<Type>::Free( void )
133 {
134     Initialize() ;
135 }
136
137
138

```

```
139 template <class Type >
140 void Sequence<Type>::Select( Predicate predicate, Sequence& output )
141 {
142     Type* item = First() ;
143     while ( item )
144     {
145         if ( predicate( *item ) )
146         {
147             output.Add( *item ) ;
148         }
149         item = Next() ;
150     }
151 }
152
153
154
155
156 #endif
157
```

```

1  /*
2  * Matías Gastón Santiago
3  * TP Programación 1 -> Prof: Gabriel Pimentel
4  *
5  * source code :: int.hpp
6  *
7  */
8
9  #ifndef INT H
10 #define INT H
11
12 struct TInt
13 {
14     //Private
15     int rep;
16
17     //Public
18     //Required to link
19     typedef int rtype;
20     typedef int stype;
21
22     bool Init( void );
23
24     stype GetValue(void);
25     bool SetValue( stype value );
26
27     static rtype Get( TInt* pint );
28     static bool Set( TInt* pint, stype value );
29     static bool Equal( TInt* pint a, TInt* pint b );
30
31     void Increment(void);
32     void Decrement(void);
33 };
34
35 void TInt::Increment(void)
36 {
37     rep++;
38 }
39
40 void TInt::Decrement(void)
41 {
42     rep--;
43 }
44
45 typename TInt::rtype TInt::Get( TInt* pint )
46 {
47     return pint->GetValue();
48 }
49
50 bool TInt::Set( TInt* pint, stype value )
51 {
52     return pint->SetValue(value);
53 }
54
55 bool TInt::Equal( TInt* pint a, TInt* pint b )
56 {
57     return ( pint a->GetValue() == pint b->GetValue() );
58 }
59
60 bool TInt::SetValue(stype value )
61 {
62     rep = value;
63     return true;
64 }
65
66
67
68
69

```

```
70  typename TInt::rtype TInt::GetValue(void)
71  {
72      return rep;
73  }
74
75  bool TInt::Init(void)
76  {
77      rep = 0;
78      return true;
79  }
80
81  #endif
82
```