```
 1  struct Data {
 2      int x, y, z;
 3  };
 4
 5  Data ** foo(Data ** v, int x) {
 6      for (int i = 0; i < x; i++)
 7          //if (v[i] != 0)
 8              v[i] = new Data;
 9      return v;
10  }
11
12  int main () {
13      const int size = 5;
14      Data ** v = new Data * [size];
15      //foo(v, size);
16      Data ** p = foo(v, size);
17      delete [] p;
18  }
19
20  /**
21  ==21913== Memcheck, a memory error detector
22  ==21913== Copyright (C) 2002-2011, and GNU GPL'd, by Julian Seward et al.
23  ==21913== Using Valgrind-3.7.0 and LibVEX; rerun with -h for copyright info
24  ==21913== Command: ./Data.out
25  ==21913==
26  ==21913==
27  ==21913== HEAP SUMMARY:
28  ==21913==     in use at exit: 100 bytes in 6 blocks
29  ==21913==   total heap usage: 6 allocs, 0 frees, 100 bytes allocated
30  ==21913==
31  ==21913== 100 (40 direct, 60 indirect) bytes in 1 blocks are definitely lost in loss record 2 of 2
32  ==21913==    at 0x4C2AC27: operator new[](unsigned long) (in /usr/lib/valgrind/vgpreload_memcheck-
    amd64-linux.so)
33  ==21913==    by 0x40060E: main (Data.cpp:14)
34  ==21913==
35  ==21913== LEAK SUMMARY:
36  ==21913==    definitely lost: 40 bytes in 1 blocks
37  ==21913==    indirectly lost: 60 bytes in 5 blocks
38  ==21913==      possibly lost: 0 bytes in 0 blocks
39  ==21913==    still reachable: 0 bytes in 0 blocks
40  ==21913==         suppressed: 0 bytes in 0 blocks
41  ==21913==
42  ==21913== For counts of detected and suppressed errors, rerun with: -v
43  ==21913== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 2 from 2)
44
45  */
46  /**
47  ==21983== Memcheck, a memory error detector
48  ==21983== Copyright (C) 2002-2011, and GNU GPL'd, by Julian Seward et al.
49  ==21983== Using Valgrind-3.7.0 and LibVEX; rerun with -h for copyright info
50  ==21983== Command: ./Data.out
51  ==21983==
52  ==21983==
53  ==21983== HEAP SUMMARY:
54  ==21983==     in use at exit: 100 bytes in 6 blocks
55  ==21983==   total heap usage: 6 allocs, 0 frees, 100 bytes allocated
56  ==21983==
57  ==21983== 100 (40 direct, 60 indirect) bytes in 1 blocks are definitely lost in loss record 2 of 2
58  ==21983==    at 0x4C2AC27: operator new[](unsigned long) (in /usr/lib/valgrind/vgpreload_memcheck-
    amd64-linux.so)
59  ==21983==    by 0x40060E: main (Data.cpp:14)
60  ==21983==
61  ==21983== LEAK SUMMARY:
62  ==21983==    definitely lost: 40 bytes in 1 blocks
63  ==21983==    indirectly lost: 60 bytes in 5 blocks
64  ==21983==      possibly lost: 0 bytes in 0 blocks
65  ==21983==    still reachable: 0 bytes in 0 blocks
66  ==21983==         suppressed: 0 bytes in 0 blocks
67  ==21983==
68  ==21983== For counts of detected and suppressed errors, rerun with: -v
69  ==21983== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 2 from 2)
70
```

```
71  */
72  /**
73  ==21994== Memcheck, a memory error detector
74  ==21994== Copyright (C) 2002-2011, and GNU GPL'd, by Julian Seward et al.
75  ==21994== Using Valgrind-3.7.0 and LibVEX; rerun with -h for copyright info
76  ==21994== Command: ./Data.out
77  ==21994==
78  ==21994==
79  ==21994== HEAP SUMMARY:
80  ==21994==     in use at exit: 60 bytes in 5 blocks
81  ==21994==   total heap usage: 6 allocs, 1 frees, 100 bytes allocated
82  ==21994==
83  ==21994== 60 bytes in 5 blocks are definitely lost in loss record 1 of 1
84  ==21994==    at 0x4C2B1C7: operator new(unsigned long) (in /usr/lib/valgrind/vgpreload_memcheck-
    amd64-linux.so)
85  ==21994==    by 0x400626: foo(Data**, int) (Data.cpp:8)
86  ==21994==    by 0x400673: main (Data.cpp:16)
87  ==21994==
88  ==21994== LEAK SUMMARY:
89  ==21994==    definitely lost: 60 bytes in 5 blocks
90  ==21994==    indirectly lost: 0 bytes in 0 blocks
91  ==21994==      possibly lost: 0 bytes in 0 blocks
92  ==21994==    still reachable: 0 bytes in 0 blocks
93  ==21994==         suppressed: 0 bytes in 0 blocks
94  ==21994==
95  ==21994== For counts of detected and suppressed errors, rerun with: -v
96  ==21994== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 2 from 2)
97
98  */
```

```
 1  Error message:
 2  In file included from cprog09lab15.cpp:8:0:
 3   vector.h: In constructor Vector<T>::Vector(size_t, T) [with T = int, size_t = unsigned int]:
 4  cprog09lab15.cpp:38:7: instantiated from void {anonymous}::VectorTester<T>::handle() [with T =
    int]
 5  cprog09lab15.cpp:198:15: instantiated from here
 6   vector.h:242:66: error: argument of type size_t (Vector<int>::)()const {aka unsigned int
    (Vector<int>::)()const} does not match size_t {aka unsigned int}
 7   vector.h:244:3: error: invalid use of member (did you forget the & ?)
 8   vector.h:247:6: error: size in array new must have integral type [-fpermissive]
 9   vector.h:247:6: error: aggregate value used where an integer was expected
10   vector.h:256:17: error: invalid use of member (did you forget the & ?)
11   vector.h: In constructor Vector<T>::Vector(size_t, T) [with T = unsigned int, size_t = unsigned
    int]:
12  cprog09lab15.cpp:38:7: instantiated from void {anonymous}::VectorTester<T>::handle() [with T =
    unsigned int]
13  cprog09lab15.cpp:201:15: instantiated from here
14   vector.h:242:66: error: argument of type size_t (Vector<unsigned int>::)()const {aka unsigned int
    (Vector<unsigned int>::)()const} does not match size_t {aka unsigned int}
15   vector.h:244:3: error: invalid use of member (did you forget the & ?)
16   vector.h:247:6: error: size in array new must have integral type [-fpermissive]
17   vector.h:247:6: error: aggregate value used where an integer was expected
18   vector.h:256:17: error: invalid use of member (did you forget the & ?)
19   vector.h: In constructor Vector<T>::Vector(size_t, T) [with T = std::basic_string<char>, size_t =
    unsigned int]:
20  cprog09lab15.cpp:38:7: instantiated from void {anonymous}::VectorTester<T>::handle() [with T =
    std::basic_string<char>]
21  cprog09lab15.cpp:204:15: instantiated from here
22   vector.h:242:66: error: argument of type size_t (Vector<std::basic_string<char> >::)()const {aka
    unsigned int (Vector<std::basic_string<char> >::)()const} does not match size_t {aka unsigned int}
23   vector.h:244:3: error: invalid use of member (did you forget the & ?)
24   vector.h:247:6: error: size in array new must have integral type [-fpermissive]
25   vector.h:247:6: error: aggregate value used where an integer was expected
26   vector.h:256:17: error: invalid use of member (did you forget the & ?)
27   vector.h: In constructor Vector<T>::Vector(size_t, T) [with T = {anonymous}::Problematic, size_t
    = unsigned int]:
28  cprog09lab15.cpp:38:7: instantiated from void {anonymous}::VectorTester<T>::handle() [with T =
    {anonymous}::Problematic]
29  cprog09lab15.cpp:207:15: instantiated from here
30   vector.h:242:66: error: argument of type size_t (Vector<{anonymous}::Problematic>::)()const {aka
    unsigned int (Vector<{anonymous}::Problematic>::)()const} does not match size_t {aka unsigned int}
31   vector.h:244:3: error: invalid use of member (did you forget the & ?)
32   vector.h:247:6: error: size in array new must have integral type [-fpermissive]
33   vector.h:247:6: error: aggregate value used where an integer was expected
34   vector.h:256:17: error: invalid use of member (did you forget the & ?)
```

plus plus . forum / forum / beginner / 18333

```
/**
%%%%%%%%%%%%%%%%%%% 1.5 Templates(mallar)

a) Modifiera din vektorklass Vector från uppgift 1.4 så att den kan
lagra en GODTYCKLIG DATATYP genom att använda mallar (templates).
Din nya klass ska dessutom KUNNA ÄNDRA STORLEK EFTER DEN SKAPATS.
Klassen ska fortfarande kasta std::out_of_range vid ogiltig åtkomst.

Exempel på instansiering:
        class A;
        ...
        Vector<double> dvect;
        Vector<A *> apvect;
        Vector<int> ivect(10);

-- Defaultkonstruktorn ska skapa en tom vektor.
Om man anger en storlek till konstruktorn ska elementen initieras till defaultvärdet för typen
(tilldela värdet T() till elementen).

-- Implementera även en konstruktor som tar två argument,
dels en storlek och ett defaultvärde för alla element i vektorn.

b) Du ska också lägga till ny funktionalitet i din klass:

% push_back(T) lägger till ett element sist i vektorn. Det ska för det mesta
  ske i konstant tid.

% insert(size_t i, T) lägger till ett element före plats i. Om i är lika
  med antal element i vektorn fungerar metoden som push_back.

% erase(size_t i) tar bort ett element på plats i

% clear() tar bort alla element

% size() ger antal element i vektorn.

% sort(bool ascending = true) sorterar vektorn i angiven riktning på
  enklast möjliga sätt. Använd std::sort (datatyper som ska jämföras måste
  definiera operator<.)

% exists(const T &) Returnerar true om elementet finns i vektorn annars
  false. Använd std::find för att implemtera funktionen.

Se till att rena åtkomstfunktioner(read only) är konstantdeklarerade.

Prova din nya vektorklass med filen test_template_vec.cpp. Detta testprogram
kontrollerar inte all funktionalitet. Du måste själv ansvara för att skriva
ett bättre testprogram som testar randvillkoren ordentligt. Ett minimum av test
är att skriva ett test för varje medlemsfunktion man implementerar.

Glöm inte använda valgrind.
*/

#ifndef VECTOR_H
#define VECTOR_H

#include <stdexcept>
//#include <cstring>
#include <iomanip>
#include <iterator>
#include <iostream>
#include <cstddef>    //size_t type is defined in the cstddef header
#include <initializer_list>
#include <cassert>
#include <algorithm>  // to call find() and sort(begin, end)

using namespace std;

template <typename T>
class Vector
{
  template <typename U>
```

```
72    friend int get_vectorCount();
73    template <typename U>
74    friend istream &operator>> (istream &, Vector<U> &);
75    template <typename U>
76    friend ostream &operator<< (ostream &, const Vector<U> &);
77
78    public:
79      /*
80      // Funktionsobjekt(Functor): less_than()
81      // Functor är objekt som imiterar syntaxen hos en funktion
82         The function-object classes that represent operators (here <) are often
83         used to override the default operator used by an algorithm (i.e. sort()).
84      */
85      struct less_than
86      {
87        bool operator()(T x, T y)
88        {
89           return x < y;
90        }
91      };
92
93  // Default constructor
94      Vector();
95
96  // Other constructor(size)
97      explicit Vector(const size_t size);
98
99  // Other constructor(size, init), initialize to default value for all elements
100     Vector(const size_t size, T );
101
102 // kopieringskonstruktor.
103     Vector(const Vector<T> &);
104
105     //init list copy constructor{1,2,3,...}
106     Vector(const initializer_list<T> &);
107
108 // Destructor
109     ~Vector();
110
111 // Assignment operator
112     const Vector<T>& operator=(const Vector<T> &);
113
114     //tilldelningsoperator som tar en initializer_list som parameter.
115     const Vector<T>& operator=(const initializer_list<T> );
116
117
118 //move-konstruktor och move-operator så att std::move fungerar.
119     Vector(Vector<T> &&) noexcept;
120
121     Vector<T>& operator=(Vector<T> &&) noexcept;
122
123
124 //överlagra indexoperatorn [] för snabb åtkomst av elementen
125     T& operator[](const int );
126
127 //operatorn [] som en konstant medlemsfunktion
128     const T& operator[](const int ) const;
129
130 // storleken
131     size_t length() const;
132
133 //======= Extra functions for test driver ========================
134
135 // Counter: return count of vectors instantiated
136    //int get_vectorCount();
137
138 // Compare equal
139    bool operator== (const Vector<T> &) const;
140
141 // Determine if 2 vectors are not equal and
142 // return true, otherwise return false (uses operator==)
143    bool operator!= (const Vector<T> &right) const
```

```
144     {
145       return ! (*this == right);
146     }
147
148     static void runTestDriver();
149
150  // ====== Ny funktionalitet i klassen  Vector ==================
151      //bool less_than (const T&, const T&);
152      void push_back(const T&);
153      void insert(const size_t i, const T& );
154      void erase(const size_t i);
155      void clear(); // ta bort alla element
156      size_t size() const;
157      void sort(bool ascending = true);
158      bool exists(const T& );
159
160    private:
161      size_t nbElements; // the logical size: nb of elements it ALREADY holds
162      size_t capacity; // the maximum possible size (max nb of elements can hold)
163
164      T *ptr;       // pointer to the first element
165      //static int vectorCount;  // # of vectors instantiated
166      int vectorCount;  // # of vectors instantiated
167      void extend_vec ();  // to expand the vector
168
169  };
170
171  // %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
172  // %%%%%%%%%%%%%%%%%%%%%%%% Cpp Implementation code %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
173
174  /*
175  A central problem in programming is the fixed size of arrays. This problem can be avoided by
     dynamically allocating vectors. The way to handle this is to make the vector bigger: Expanding
     the vector.
176
177  Geometric expansion and amortized cost:
178  To avoid incurring the cost of resizing many times, dynamic arrays resize by a large amount, such
     as doubling in size, and use the reserved space for future expansion.
179  */
180
181  // resize vector (Ridimensionamento)
182  // Strategy of DOUBLING of the current capacity
183  template <typename T>
184    void Vector<T>::extend_vec() //(const size_t sz)
185  {
186    if (nbElements>capacity/2)
187    {
188       // capacity is the real size of the allocated memory
189       capacity *= 2;  // double the previous allocated memory   capacity = nbElements * 2 ;
190
191       T *temp = ptr;
192       ptr = new T[capacity]; // create the new extended vector(memory allocation)
193
194       for (size_t i=0; i<nbElements; i++)  // copy the elements from the old memory
195         ptr[i] = temp[i];
196
197       delete[] temp; // deallocation of the old memory
198    }
199  }
200
201  // # Old Definitions
202
203  //======================================== DEFAULT CONSTRUCTOR
204  // Defaultkonstruktorn ska skapa en tom vektor.
205  template <typename T>
206    Vector<T>::Vector()
207  {
208    nbElements = 0;  //nb of elements it ALREADY holds
209    capacity = 0;    // the max nb of elements can hold
210    ptr = nullptr;
211    ++vectorCount;   // count one more object
212  }
```

```
213
214  //========================================== OTHER CONSRUCTOR
215  //Om man anger en storlek till konstruktorn ska elementen
216  //initieras till defaultvärdet för typen (tilldela värdet T() till elementen).
217  // Other constructor(size, initialize to t())
218  template <typename T>
219    Vector<T>::Vector(const size_t size) : capacity(size)
220  {
221    if (capacity > 0)
222    {
223       nbElements = 0;
224       ptr = new T[size];
225       //extend_vec(size);  // Expand the vector to a new size
226       //size = capacity;   // resize the vector
227
228       ++vectorCount;     // count one more object
229    }
230    else
231      ptr = nullptr;
232
233    for (size_t i=0; i<size; i++) //for (auto i : v) // C++11
234      ptr[i] = T();               //elementen initieras till defaultvärdet för T
235  }
236
237  //==================================== CONSTRUCTOR WITH 2 ARGUMENTS
238  // Other constructor(size, init)
239  // initialize all elements to a default value
240
241  template <typename T>
242    Vector<T>::Vector(const size_t size, T init) : capacity(size)
243  {
244    if (capacity > 0)
245    {
246       nbElements = 0;
247       ptr = new T[size];
248       //extend_vec(size);  // Expand the vector to a new size
249       //size = capacity;   // resize the vector
250
251       ++vectorCount;     // count one more object
252
253       for (size_t i=0; i<size; i++) //for (auto i : v) // C++11
254       {
255         ptr[i] = init;              //elementen initieras till ett defaultvärdet
256         nbElements++;
257       }
258       extend_vec();
259    }
260    else
261      ptr = nullptr;
262
263    //for (size_t i=0; i<size; i++) //for (auto i : v) // C++11
264     // ptr[i] = init;              //elementen initieras till ett defaultvärdet
265  }
266
267  // ============================================== COPY CONSTRUCTOR
268
269  /*
270     # E' necessario creare esplicitamente un costruttore di
271       copia quando un oggetto ha dati membro allocati
272       dinamicamente.(in questo caso ptr=new unsigned int[size])
273     # Per oggetti con dati allocati dinamicamente la copia bit-a-
274       bit copia solo il dato puntatore, cioè l'indirizzo di
275       memoria
276  */
277  // Copy constructor: MUST receive a reference to prevent
278  // infinite recursion(Dipendenza logica circolare)
279  // Advice: use constructor initializers!
280  template <typename T>
281    Vector<T>::Vector(const Vector<T> &copia) : nbElements(copia.nbElements)
282  {
283    capacity = nbElements;
284
```

```
285     if (capacity > 0)
286     {
287       //extend_vec();
288       ptr = new T[capacity];
289       //extend_vec(size);  // Expand the vector to a new size
290       //size = capacity;   // resize the vector
291
292       ++vectorCount;                  // count one more object
293
294       for (size_t i=0; i<nbElements; i++)
295       {
296         ptr[i] = copia.ptr[i];        //copy copia into object
297         //nbElements++;
298       }
299       //extend_vec();     // expand vector if it is necessary
300     }
301     else ptr = nullptr;
302   }
303
304   //  ========================================== EXTRA COPY CONSTRUCTOR
305   // initializer_list copy constructor: {1,2,3,...}
306   template <typename T>
307     Vector<T>::Vector(const initializer_list<T> &il) : capacity(il.size())
308   {
309     nbElements = capacity;
310     if (capacity > 0)
311     {
312
313        extend_vec();  // Expand the vector if it is necessary
314        //size = capacity;   // resize the vector
315        ptr = new T[capacity];
316        ++vectorCount;
317
318     /**
319     begin and end return iterators of type
320     initializer_list<size_t>::iterator
321     c++11 standard: auto iter = il.begin()
322     */
323        //typename initializer_list<T>::iterator iter;
324        int i = 0;
325        for (auto iter=il.begin(); iter!=il.end(); ++iter)
326        {
327           ptr[i++] = *iter;
328           //nbElements++;
329        }
330        //extend_vec();     // Expand vector if it is necessary
331     }
332
333     else ptr = nullptr;
334   }
335
336   // ======================================================== DESTRUCTOR
337   template <typename T>
338     Vector<T>::~Vector()
339   {
340     delete[] ptr;
341     --vectorCount;    // one fewer object
342   }
343
344   // OVERLOADING ============================================================
345   /**
346   ############    Overloading di Operatori
347
348   • Perché un operatore possa operare su gli oggetti di
349     una classe deve necessariamente essere ridefinito
350   • fanno eccezione:
351      - operatore di assegnamento =
352        il suo comportamento di default è di eseguire una copia di
353        membro a membro dei dati di una classe
354      - operatore di indirizzo &
355        il suo comportamento di default è di restituire l'indirizzo di
356        memoria dell'oggetto
```

```
357   • tuttavia anche = e & possono essere ridefiniti se
358     serve
359   */
360   // ========================================= ASSIGNMENT OPERATOR
361   // const return avoids: (a1 = a2) = a3
362   template <typename T>
363     const Vector<T> &Vector<T>::operator=(const Vector<T> &right)
364   {
365     if (&right == this)   //check for self-assignment
366         return *this;     //same object
367
368     //for vector of different sizes, deallocate original
369     //left side vector then allocate new left side vector
370
371     else if (capacity != right.capacity)
372     {
373        delete [] ptr;  // clear();
374        capacity = right.capacity;
375       nbElements = 0;
376
377        //extend_vec(size);  // Expand the vector to a new size
378        //size = capacity;   // resize the vector
379        ptr = new T[capacity];
380       assert(ptr != nullptr);    //terminate if memory not allocated
381     }
382     for (size_t i=0; i<capacity; i++)
383     {
384        ptr[i] = right.ptr[i];
385       nbElements++;
386     }
387     //extend_vec();
388
389     return *this;
390   }
391
392   //  ======================================== ASSIGNMENT {x1,x2,x3...}
393   //tilldelningsoperator som tar en initializer_list som parameter.
394   template <typename T>
395     const Vector<T> &Vector<T>::operator=(const initializer_list<T> rightlist)
396   {
397     if (ptr != nullptr)
398       delete [] ptr;  //clear();
399
400     capacity = rightlist.size();
401     nbElements = 0;
402     //extend_vec(size);  // Expand the vector to a new size
403     //size = capacity;   // resize the vector
404
405     //typename initializer_list<T>::iterator iter;
406     int i = 0;
407     for (auto it=rightlist.begin(); it!=rightlist.end(); ++it)
408     {
409       ptr[i++] = *it;
410       nbElements++;
411     }
412     //extend_vec();
413
414     return *this;
415   }
416
417   //================================================= MOVE CONSTRUCTOR
418   /**
419   Move semantics enables you to write code that transfers, "steal", resources
420   (such as dynamically allocated memory) from one object to another.
421    Move semantics works because it enables resources to be transferred
422   from temporary objects that cannot be referenced elsewhere in the program.
423   To implement move semantics, you typically provide a move constructor,
424   and optionally a move assignment operator (operator=), to your class.
425   Unlike the default copy constructor, the compiler does not provide a
426   default move constructor.
427   By "noexcept" we explicitly tell the library that our move constructor
428   is safe to use in circumstances such as vector reallocation.
```

```
429   */
430   //move-konstruktor:operates by "moving" resources from
431   //the given object to the object being constructed
432   template <typename T>
433     Vector<T>::Vector(Vector<T> &&tomove) noexcept
434   {
435     ptr = tomove.ptr;      //assign the class data members from the source
436     capacity = tomove.capacity; //object to the object that is being constructed
437     nbElements = tomove.nbElements;
438
439     //extend_vec();
440
441     tomove.ptr = nullptr;  //Assign the data members of the source object to
442     tomove.capacity = 0;   //default values. This prevents the destructor from
443                            //freeing resources (such as memory) multiple times.
444   }
445
446   //================================================== MOVE-OPERATOR
447   //move-operator så att std::move fungerar
448   template <typename T>
449     Vector<T> &Vector<T>::operator=(Vector<T> &&source) noexcept
450   {
451     if (this == &source)       //Self-assignment
452       return *this;
453
454     if (ptr != nullptr)
455       delete[] ptr;
456
457     ptr = source.ptr;           // Copy the vector pointer and its
458     capacity = source.capacity; // size from the source object.
459     nbElements = source.nbElements;
460
461     //extend_vec();
462
463
464     source.ptr = nullptr;      // Release the data members from the source
465     source.capacity = 0;       // object so that the destructor does not
466                                // free the memory multiple times.
467
468     return *this;              // Return a reference to the current object
469   }
470
471   // ================================================== INDEX OPERATOR
472   //överlagra indexoperatorn [] för snabb åtkomst av elementen
473   // Overloaded subscript operator for non-const vectors
474   // reference return creates an lvalue
475   template <typename T>
476     T &Vector<T>::operator[](const int subscript)
477   {
478     // check for subscript out of range error
479     if ((subscript<0)||(subscript≥(int)capacity-1))
480       throw out_of_range(" Out of range !");
481
482     return ptr[subscript]; // reference return
483   }
484
485   // ================================================== INDEX OPERATOR 2
486   //operatorn [] som en konstant medlemsfunktion
487   // Overloaded subscript operator for const vectors
488   // const reference return creates an rvalue
489   template <typename T>
490     const T& Vector<T>::operator[](const int subscript) const
491   {
492     // check for subscript out of range error
493     if ((subscript<0)||(subscript≥(int)capacity-1))
494       throw out_of_range(" Out of range !!");
495
496     return ptr[subscript]; // const reference return
497   }
498
499   //%%%%%%%%%% Extra functions utility %%%%%%%%%%%%%%%%%%%%%%%%%%%%
500
```

```
501  // Return number of vector objects instantiated
502  // static functions cannot be const
503  template <typename U>
504    int get_vectorCount()              // friend function
505    //int Vector<T>::get_vectorCount()
506  {
507    return Vector<U>::vectorCount;
508  }
509
510  //================================================
511    // Overload input operator for class vector;
512    // inputs values for entire vector.
513  template <typename U>
514    istream &operator>> (istream &input, Vector<U> &v)  // friend function
515  {
516    for (unsigned int i=0; i<v.capacity; i++)
517    {
518      input >> v.ptr[i];
519      v.nbElements++;
520    }
521   // v.extend_vec();
522
523    return input;   // enables cin >> x >> y;
524  }
525
526  //================================================
527  // Overload output operator for class vector
528  template <typename U>
529    ostream &operator<< (ostream &output, const Vector<U> &v)  // friend function
530  {
531    unsigned int i;
532    for (i=0; i<v.nbElements; i++)
533    {
534       output << setw(5) << v.ptr[i];
535       if ((i+1) % 3 == 0)  // 3 elements per row
536         output << endl;
537    }
538
539    if (i%3 != 0)
540      output << endl;
541
542    return output;    // enables cout << x << y;
543  }
544
545  //================================================
546  // Determine if 2 vectors are equal and
547  // return true, otherwise return false.
548  template<typename T>
549    bool Vector<T>::operator== (const Vector<T> &right) const
550  {
551    if (nbElements != right.nbElements)
552      return false;       // vectors of different sizes
553
554    for (unsigned int i=0; i<capacity; i++)
555      if (ptr[i] != right.ptr[i])
556        return false;     // vectors are not equal
557
558    return true;          // vectors are equal
559  }
560
561  //%%%%%%%%%%%%%%%%%% New Definitions %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
562  //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
563
564  // ================================================== push_back(T)
565  // lägger till ett element sist i vektorn.
566  // Det ska för det mesta ske i konstant tid.
567  /*
568  Every implementation is required to follow a strategy that ensures that it is
569  efficient to use push_back to add elements to a vector.
570  The execution time of creating an n-element vector by calling push_back n times
571  on an initially empty vector must never be more than a CONSTANT MULTIPLE  of n.
572
```

```
573  This effectively increases the vector size by one, which causes a reallocation
574  of the internal allocated storage if the vector size was equal to the vector
575  capacity before the call.
576  */
577  template <typename T>
578    void Vector<T>::push_back(const T& element)
579  {
580
581    if (capacity==0)
582    {
583       capacity = 1;
584       nbElements = 0;
585       ptr = new T[capacity];
586       ptr[0] = element;
587       nbElements++;
588    }
589    else if (nbElements==capacity)
590    {
591       extend_vec();
592       //assert (nbElements<capacity);
593       ptr[nbElements] = element;
594       nbElements++;
595    }
596
597    //extend_vec();
598  }
599
600  //============================================================ insert()
601  // insert(size_t i, T) lägger till ett element före plats i. Om i är lika
602  // med antal element i vektorn fungerar metoden som push_back.
603  template <typename T>
604    void Vector<T>::insert(const size_t position, const T& element)
605  {
606    //assert (nbElements<capacity);
607    if ((position<0) || (position>nbElements))
608      throw std::out_of_range(" Out of range !");
609
610    if (capacity == nbElements)
611      extend_vec();
612
613    for (size_t i=nbElements; i>position; i--)
614    {
615       ptr[i] = ptr[i-1];   // Decale les elements vers le haut
616    }
617
618    ptr[position] = element;
619    nbElements++;
620
621    //extend_vec();
622  }
623
624  //============================================================ erase()
625  // erase(size_t i) tar bort ett element på plats i
626  template <typename T>
627    void Vector<T>::erase(const size_t position)
628  {
629     assert (nbElements != 0);
630     if ((position<0) || (position>=nbElements))
631        throw std::out_of_range(" Out of range !");
632
633     nbElements--;
634     for (size_t i=position; i<nbElements; i++)
635    {
636       ptr[i] = ptr[i+1];   // Decale les elements vers le bas
637    }
638    //nbElements--;
639  }
640
641  //============================================================ clear()
642  //clear() tar bort alla element
643  template <typename T>
644    void Vector<T>::clear()
```

```
645   {
646     if (ptr != nullptr)
647     {
648       delete [] ptr;
649       ptr = nullptr;
650     }
651     nbElements = capacity = 0;
652   }
653
654   //============================================================ size()
655   // size() ger antal element i vektorn.
656   template <typename T>
657     size_t Vector<T>::size() const
658   {
659       return nbElements;
660   }
661
662   //============================================================ sort()
663   /*
664   A call to sort() arranges the elements in the input range into sorted order using the element
      type's < operator.
665   sort() is not stable: equivalent elements that are ordered one way before sorting may be ordered
      differently after sorting.
666   */
667   // sort(bool ascending = true) sorterar vektorn i angiven riktning på
668   // enklast möjliga sätt. Använd std::sort (datatyper som ska jämföras måste
669   // definiera operator<.)
670   template <typename T>
671     void Vector<T>::sort(const bool ascending)
672   {
673     if (nbElements < 2)
674       return;
675
676     //std::sort(&ptr[0], &ptr[nbElements]);  // call to std::sort()
677
678     //bool (*foo)(const T&, const T&);   // function pointer declaration
679     //foo = less_than;
680     //std::sort(&ptr[0], &ptr[nbElements], foo);
681
682     //std::sort(&ptr[0], &ptr[nbElements], less_than(const T&, const T& )); // why didn't work ?
683   // std::sort(&ptr[0], &ptr[nbElements], less_than()); // functor less_than()
684     // Overloaded version of sort()
685     //this version of sort takes a third argument that is a predicate
686
687     if (ascending == false)
688     {
689        T* temp = new T[capacity];
690
691        for (size_t i=0; i<nbElements; i++)
692          temp[nbElements-i-1] = ptr[i];
693
694        delete [] ptr;
695        ptr = temp;
696     }
697
698   }
699
700   //============================================================ exists()
701   // exists(const T &) Returnerar true om elementet finns i vektorn annars
702   // false. Använd std::find för att implemtera funktionen.
703   template <typename T>
704     bool Vector<T>::exists(const T& val)
705   {
706     if (nbElements == 0)
707       return 0;
708
709     T* result = find(&ptr[0], &ptr[nbElements], val);
710
711     return result != ptr[nbElements];
712   }
713   /*
714   //====================== Auxilliary function to sort(),less_than()
```

```
715  template <typename T>
716    bool Vector<T>::less_than(const T& x, const T& y)
717  {
718      return (x < y);
719  }
720  */
721
722
723  #endif
```

```
//1.3  Tempor objekt, minnesl or, valgrind
#include <iostream>

class A {
public:
    A()
        {std::cout << "The default contructor" << std::endl; }
    A(const A & ref)
        {std::cout << "The copy contructor" << std::endl; }
    ~A()
        {std::cout << "The destructor" << std::endl; }
    A(char * s)
        {std::cout << "Some other constructor " << s << std::endl;}
    A & operator=(const A & s)
        {std::cout << "The assignment operator" << std::endl;
         return *this;}
};

void no_ref(A a) {}
void with_ref(const A & a) {}

int main()
{
  A a("my name is a");
  A b = a;          // vad skillnaden
  A c(a);           // mellan dessa
  A d;              // tre tekniker?
  d = a;

  no_ref(a);        // Bildas tempor objekt?
  with_ref(a);      // Bildas tempor objekt?

  A *aa = new A[5];
  delete[]aa;       // Vad kommer att h a?
  return 0;
}
/*
==7186== Memcheck, a memory error detector
==7186== Copyright (C) 2002-2011, and GNU GPL'd, by Julian Seward et al.
==7186== Using Valgrind-3.7.0 and LibVEX; rerun with -h for copyright info
==7186== Command: ./A.out
==7186==
Some other constructor my name is a
The copy contructor
The copy contructor
The default contructor
The assignment operator
The copy contructor
The destructor
The default contructor
The default contructor
The default contructor
The default contructor
The default contructor
The destructor
==7186== Invalid free() / delete / delete[] / realloc()
```

```
==7186==    at 0x4C2A4BC: operator delete(void*) (in /usr/lib/valgrind/vgpreloa
d_memcheck-amd64-linux.so)
==7186==    by 0x400AA2: main (A.cpp:33)
==7186==  Address 0x59ff048 is 8 bytes inside a block of size 13 alloc'd
==7186==    at 0x4C2AC27: operator new[](unsigned long) (in /usr/lib/valgrind/v
gpreload_memcheck-amd64-linux.so)
==7186==    by 0x400A4D: main (A.cpp:32)
==7186==
The destructor
The destructor
The destructor
The destructor
==7186==
==7186== HEAP SUMMARY:
==7186==     in use at exit: 13 bytes in 1 blocks
==7186==   total heap usage: 1 allocs, 1 frees, 13 bytes allocated
==7186==
==7186== 13 bytes in 1 blocks are definitely lost in loss record 1 of 1
==7186==    at 0x4C2AC27: operator new[](unsigned long) (in /usr/lib/valgrind/v
gpreload_memcheck-amd64-linux.so)
==7186==    by 0x400A4D: main (A.cpp:32)
==7186==
==7186== LEAK SUMMARY:
==7186==    definitely lost: 13 bytes in 1 blocks
==7186==    indirectly lost: 0 bytes in 0 blocks
==7186==      possibly lost: 0 bytes in 0 blocks
==7186==    still reachable: 0 bytes in 0 blocks
==7186==         suppressed: 0 bytes in 0 blocks
==7186==
==7186== For counts of detected and suppressed errors, rerun with: -v
==7186== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 2 from 2)

*/
```

```
1  ////////////////////////////////////////////////////////////////
2  //
3  // Personuppgifter (namn, pnr, epost) på dem som gjort labben
4  //
5
6  ////////////////////////////////////////////////////////////////
7  //
8  // 1.1
9  //
10 // vad betyder \$* i en makefile?
11
12 När ett % tecken är med i beroendelistan (alltså t.ex %.cpp: innan kommandoraden)
13 så använder man $* för att använda den matchade texten i kommandoraden.
14
15 // vad gör -Wall och -g ?
16
17 -Wall aktiverar nästan alla varningar vid kompilering.
18 -g gör att det går att debugga den kompilerade filen.
19
20 ////////////////////////////////////////////////////////////////
21 //
22 // 1.2 a)
23 //
24 // int powerof(int x, int y) {
25 //     int res = 1;
26 //     for (int i = 0; i < y; i++); {
27 //         res *= x;
28 //     }
29 //     return res;
30 // }
31 //
32 // int main() {
33 //     int x = 10;
34 //     int y = 3;
35 //
36 //     int res = powerof(x, y);
37 //
38 //     std::cout << x << " upphöjt till " << y << " är " << res << std::endl;
39 //
40 //     float z = 0.29;
41 //     int w = (int) (z * x * x);
42 //     if (z * x * x == 29)
43 //         std::cout << z << "*" << x * x << " är 29" << std::endl;
44 //     else
45 //         std::cout << z << "*" << x * x << " är inte 29" << std::endl;
46 // }
47 //
48 // Varför blir värdet på variabeln w inte det man tror (0.29*100)?
49
50 float z= 0.29; sätter egentligen z=0.28999992 då 0.29 inte kan representeras.
51 Detta multipliceras sedan med x*x och trunkeras vilket innebär att decimalerna skärs
52 bort utan avrundning. Detta leder till att z*x*x=28.
53
54 // Hur många varv körs for-loopen i funktionen powerof?
55
56 i=0, i<y=3; i++ --> i=0,1,2. Alltså tre gånger.Avslutas med att i=3.
57 Eftersom det finns ett ; avslut efter for definitionen så är inte
58 res*=x en del av loopen. Detta gör att loopen antagligen optimeras bort
59 och i=y direkt.
60
61 // 1.2 b)
62 //
63 // int must_follow_a(char * start, int length, char a, char b) {
64 //     int nr = 0;
65 //     for (int i = 0; i < length; i++, ++start) {
66 //         if (*start == a && *(start + 1) == b) // maintainers note: DANGER!
67 //             nr += 1;
68 //     }
69 //     return nr;
70 // }
71 //
72 // Dina tre testfall
```

```
73
74     void test_b_is_second_to_last( void )
75     {//ingen teckenföljd
76       char vek[] = {'b','b','b','b','a'};
77       int result = must_follow_a(vek,5,'a','b');
78       TS_ASSERT_EQUALS( result, 0);
79     }
80
81      void test_c_is_second_to_last( void )
82     {//ingen teckenföljd
83           char vek[] = {'b', 'a', 'a', 'a', 'b'};
84           int result = must_follow_a(vek, 4, 'a', 'b');
85           TS_ASSERT_EQUALS( result, 0);
86       }
87
88     void test_d_is_second_to_last( void )
89     { //1 teckenföljd
90       char vek[] = {'b','b','a','b','a'};
91       int result = must_follow_a(vek,5,'a','b');
92       TS_ASSERT_EQUALS( result, 1);
93     }
94
95   // Varför är det så viktigt att testa randvillkoren?
96
97   Här är det lätt att missa då man läser utanför gränserna för ett minnesblock
98   eller vad man har definerat som gränser inom ett minnesblock och detta kan
99   ge error ibland och ibland inte vilket gör att det kan vara svårt att upptäcka
100  fel av denna karaktären.
101  I uppgift 1.2b borde funktionen must_follow_a inte gå till i<length och titta
102  som sista utan den borde stanna 1 tecken innan.
103
104  ////////////////////////////////////////////////////////////////
105  //
106  // 1.3
107  //
108  // Bifoga källkoden till din version av A.cpp
109
110  #include <iostream>
111
112  class A {
113  public:
114      A()
115          {std::cout << "The default contructor" << std::endl; }
116      A(const A & ref)
117          {std::cout << "The copy contructor" << std::endl; }
118      ~A()
119          {std::cout << "The destructor" << std::endl; }
120      A(char * s)
121          {std::cout << "Some other constructor " << s << std::endl;}
122      A & operator=(const A & s)
123          {std::cout << "The assignment operator" << std::endl;
124           return *this;}
125  };
126
127  void no_ref(A a) {}
128  void with_ref(const A & a) {}
129
130  int main()
131  {
132      A a("my name is a"); //  "Some other constructor" som tar char*
133      A b = a;          // vad är skillnaden. COPY constructor    (objektet a)
134      A c(a);           // mellan dessa.      COPY constructor    (objektet a)
135      A d;              // tre tekniker?      DEFAULT constructor (inga argument)
136      d = a;            //                    ASSIGNMENT operator
137
138      no_ref(a);        // Bildas temporära objekt? JA och det förstörs efter programmet går ur
     scopet.
139      with_ref(a);      // Bildas temporära objekt? NEJ. Vi skickar en referens direkt till objektet.
140
141      A *aa = new A[5]; // DEFAULT constructor*5
142      delete aa;        // Vad kommer att hända? Error eftersom den är allokerad med "new []"
143                        //ska den tas bort med "delete []" (INVALID HEAP ARGUMENT enl DrMemory)
```

```
144       return 0;
145   }
146
147
148   // Vad skriver ditt program ut, var förberedd att förklara varför.
149
150   Some other constructor my name is a
151   The copy contructor
152   The copy contructor
153   The default contructor
154   The assignment operator
155   The copy contructor
156   The destructor
157   The default contructor
158   The default contructor
159   The default contructor
160   The default contructor
161   The default contructor
162   The destructor
163   The destructor
164   The destructor
165   The destructor
166   The destructor
167
168   // När frigörs objekten?
169
170   I slutet där det står "The destructor" frigörs först den dynamiska objektinstansen aa[0]
171   och sedan de statiska objektinstanserna a,b,c,d. Kvar är då aa[1],aa[2],aa[3],aa[4].
172
173   // När skapas temporära objekt?
174
175   //    A b = a;          // vad är skillnaden
176   //    A c(a);           // mellan dessa
177   //    A d;              // tre tekniker?
178
179   A b = a; använder COPY constructor A(a); för att skapa instansen.
180   A c(a);  använder COPY constructor A(a); för att skapa instansen.
181   A d;     använder DEFAULT constructor A(); för att skapa instansen.
182
183   Alltså blir den första optimerad av kompilatorn att istället för
184   att skapa objekt b och sen assign =a så skapas det direkt med COPY konstruktorn.
185
186   //    no_ref(a);        // Bildas temporära objekt?
187
188   Ja det bildas ett temporärt objekt som förstörs när programpekaren går ur scopet.
189
190   //    with_ref(a);      // Bildas temporära objekt?
191
192   Nej det skickas en referens direkt till objektinstansen a.
193
194   //    delete aa;        // Vad kommer att hända
195
196   Eftersom den är allokerad med "new []" ska den tas bort med
197    "delete []" (INVALID HEAP ARGUMENT enl DrMemory (typ som valgrind))
198   Resultatet blir som att man gjort delete aa[0] varvid alla de andra elementen finns kvar.
199
200   ////////////////////////////////////////////////////////////////
201   //
202   // struct Data {
203   //     int x, y, z;
204   // };
205   //
206   // Data ** foo(Data ** v, int x) {
207   //     for (int i = 0; i < x; i++)
208   //         //if (v[i] != 0)
209   //             v[i] = new Data;
210   //     return v;
211   // }
212   //
213   // int main () {
214   //     const int size = 5;
215   //     Data ** v = new Data * [size];
```

```
216  //      Data ** p = foo(v, size);
217  //      delete [] p;
218  // }
219
220
221  // Hur ser valgrinds felmeddelande ut?
222
223  Uninitialized read
224  80b leak
225
226  // Blir det någon skillnad i hur mycket minne som läcker när man
227  // kommenterar if-satsen?
228
229  Nej.
230
231  // Borde det ha blivit någon skillnad?
232
233  Kanske, det beror på om minnet innehållt 0 eller inte innan
234  vi "bokade upp minnet". Det avgör om vi kommer skapa nya objekt i den första versionen.
235
236  // Varför läcker programmet fortfarande minne?
237
238  Objekt instanserna blir ej borttagna. Vi tar bara bort pekar-strukturen v.
239
240  ////////////////////////////////////////////////////////////////
241  //
242  // 1.4
243  //
244  // Generellt är det ofta en god idé att låta konstruktorer som
245  // tar ett argument deklareras som explicit. Varför? Ange ett
246  // exempel där det annars kan bli dumt.
247
248
249
250  Om en konstruktor inte är deklarerad som explicit så kan
251  kompilatorn använda den för implicit konversation mellan typer.
252
253  class databas
254  {
255  public:
256          databas(int i) { hemliganummret=i; }
257  private:
258          int hemliganumret;
259          char** rad;
260  }
261
262  void skriv_ut_databas_rad(const databas& db)
263  {
264          cout << db.rad[70];
265  }
266  int main()
267  {
268          skriv_ut_databas_rad(32);
269          //Detta går att göra eftersom klassen databas
270          //har en konstruktor som tar en int - implicit konvertering.
271  }
272
273  om databas(int i) deklareras "explicit databas(int i)" så
274  kan inte kompilatorn göra denna konvertering och ger ett Error istället.
275
276
277  // operatorn[] måste vara en konstant medlemsfunktion i vissa
278  // fall. När och varför? Hur kopierar man vektorn?
279
280  När objektet är const så kallas const versionen av operator[],
281  alltså om värdet är ett rvalue. om det är ett lvalue så anropas versionen som ej är const.
```

hnnemati@kth.se

– 4 olika classdiagram (Polymorfi): Date, Gregorian, julian, roman ¦ calendar

Grego tropical year $365\ 97/400$ 97 leap years every 400 years
week_day()
{return floor (days_from_zero + 0,5) % days_per_week()) }

floor (JD + 0,5)

MJD [2,400,000.5] ∠ JD

MJD 0 started on 7 nov 1858 (Gregorian) at 00:00:00 TT

Integer division C++ : 14 / 5

Modulo C++ : 14 % 5

2440587,5 + (K_time(0)/86400)

4713
2000
6712

1º gennaio 4713 B.C. ⟶ JD.

1 Julian period = 7980 year from 1 january 4713.

4713
1970
6683

bool is_date (int y, int m, int d) ;

Dagens datum ; 2440587,5 (JD number of 1 jan 1970) + $\left(\frac{K\_time(0)}{86400}\right)$ → time

january 1, 1970

(1970 ← 4713 BC) = (4713 + 1970) = 6683 years = 6682

6683 −1 × 365,25 = 2441000 [2440587] Anno

40587

$2440587 + \frac{K\_time(0)}{86400}$

1m → 60 sec
1h → 60 min
1day → 24 h.
24 × 60 × 60 = 86400

(1972 jan 05) + 1 month.

(26) + 1 → Feb 1

550 + (~~1000~~ - 5) = 555

519

to transform time to Day

```
// projektlokala headerfiler
#include "kattistime.h"
#include "julian.h"
#include "gregorian.h"

// STL headerfiler
#include <iostream>
#include <assert.h>                // assert(b) ger felmeddelande om b falsk
#include <ctime>


// Obs att testerna f
```

- worth_value.

- lab22a.in
  lab22a.out

  lab22c.in
  lab22c.out

  lab23.in
  lab23.out.

  set_x_time(time)

  - kompilera med g++ -o lab22b → result.out
  - xor result.out med lab22c.in

-

```
g++ -g -std=c++0x -Wall -c cprog09lab22a.cpp date.cpp roman.cpp julian.cpp kattistime.cpp
In file included from cprog09lab22a.cpp:8:0:
julian.h:78:12: error: invalid abstract return type for member function 'lab2::Julian
lab2::Julian::operator++(int)'
julian.h:40:9: note:   because the following virtual functions are pure within 'lab2::Julian':
date.h:102:25: note:    virtual std::string lab2::Date::month_name() const
julian.h:79:12: error: invalid abstract return type for member function 'lab2::Julian
lab2::Julian::operator--(int)'
julian.h:40:9: note:   since type 'lab2::Julian' has pure virtual functions
cprog09lab22a.cpp: In function 'void {anonymous}::new_date() [with D = lab2::Julian]':
cprog09lab22a.cpp:86:21:   instantiated from here
cprog09lab22a.cpp:58:5: error: cannot allocate an object of abstract type 'lab2::Julian'
julian.h:40:9: note:   since type 'lab2::Julian' has pure virtual functions
cprog09lab22a.cpp:60:5: error: cannot allocate an object of abstract type 'lab2::Julian'
julian.h:40:9: note:   since type 'lab2::Julian' has pure virtual functions
cprog09lab22a.cpp: In function 'void {anonymous}::new_date_copy() [with D = lab2::Julian]':
cprog09lab22a.cpp:88:26:   instantiated from here
cprog09lab22a.cpp:71:4: error: cannot allocate an object of abstract type 'lab2::Julian'
julian.h:40:9: note:   since type 'lab2::Julian' has pure virtual functions
In file included from julian.cpp:11:0:
julian.h:78:12: error: invalid abstract return type for member function 'lab2::Julian
lab2::Julian::operator++(int)'
julian.h:40:9: note:   because the following virtual functions are pure within 'lab2::Julian':
date.h:102:25: note:    virtual std::string lab2::Date::month_name() const
julian.h:79:12: error: invalid abstract return type for member function 'lab2::Julian
lab2::Julian::operator--(int)'
julian.h:40:9: note:   since type 'lab2::Julian' has pure virtual functions
julian.cpp: In static member function 'static void lab2::Julian::runTestDriver()':
julian.cpp:24:13: error: cannot declare variable 'j' to be of abstract type 'lab2::Julian'
julian.h:40:9: note:   since type 'lab2::Julian' has pure virtual functions
julian.cpp: In member function 'lab2::Julian lab2::Julian::operator++(int)':
julian.cpp:115:12: error: invalid abstract return type for member function 'lab2::Julian
lab2::Julian::operator++(int)'
julian.h:40:9: note:   since type 'lab2::Julian' has pure virtual functions
julian.cpp:117:15: error: cannot declare variable 'source' to be of abstract type 'lab2::Julian'
julian.h:40:9: note:   since type 'lab2::Julian' has pure virtual functions
julian.cpp:119:15: error: cannot allocate an object of abstract type 'lab2::Julian'
julian.h:40:9: note:   since type 'lab2::Julian' has pure virtual functions
julian.cpp: In member function 'lab2::Julian lab2::Julian::operator--(int)':
julian.cpp:122:12: error: invalid abstract return type for member function 'lab2::Julian
lab2::Julian::operator--(int)'
julian.h:40:9: note:   since type 'lab2::Julian' has pure virtual functions
julian.cpp:124:15: error: cannot declare variable 'source' to be of abstract type 'lab2::Julian'
julian.h:40:9: note:   since type 'lab2::Julian' has pure virtual functions
julian.cpp:126:15: error: cannot allocate an object of abstract type 'lab2::Julian'
julian.h:40:9: note:   since type 'lab2::Julian' has pure virtual functions
make: *** [testjulian] Error 1
```

you cannot use abstract class as a fct. return type. But you can return pointer or reference to it Like in following declarations!

BodyNode * getNext( ) ;

Or BodyNode& getNext( ) ;

class class

calendar.h :98:5 error: 'const lab2::Calendar<lab2::Gregorian>'
has no member named 'get_eventlist'

```
red-19:~/c++prog/labbar/laboration2>make testcalendar
g++ -g -std=c++0x -Wall -c cprog09lab23.cpp date.cpp roman.cpp julian.cpp gregorian.cpp
kattistime.cpp
calendar.h: In copy constructor 'lab2::Calendar<T>::Calendar(const lab2::Calendar<T>&) [with T =
lab2::Gregorian]':
cprog09lab23.cpp:68:6:   instantiated from 'void {anonymous}::CalendarTester<T, S>::handle() [with
T = lab2::Gregorian, S = lab2::Julian]'
cprog09lab23.cpp:175:13:   instantiated from here
calendar.h:98:5: error: 'const class lab2::Calendar<lab2::Gregorian>' has no member named
'get_eventlist'
calendar.h: In constructor 'lab2::Calendar<T>::Calendar(const lab2::Calendar<U>&) [with U =
lab2::Julian, T = lab2::Gregorian]':
cprog09lab23.cpp:71:6:   instantiated from 'void {anonymous}::CalendarTester<T, S>::handle() [with
T = lab2::Gregorian, S = lab2::Julian]'
cprog09lab23.cpp:175:13:   instantiated from here
calendar.h:162:5: error: 'const class lab2::Calendar<lab2::Julian>' has no member named
'get_eventlist'
calendar.h:83:29: error: 'std::list<lab2::EventDate<lab2::Julian>,
std::allocator<lab2::EventDate<lab2::Julian> > > lab2::Calendar<lab2::Julian>::eventlist' is
private
calendar.h:166:42: error: within this context
cprog09lab23.cpp:71:6:   instantiated from 'void {anonymous}::CalendarTester<T, S>::handle() [with
T = lab2::Gregorian, S = lab2::Julian]'
cprog09lab23.cpp:175:13:   instantiated from here
calendar.h:83:29: error: 'std::list<lab2::EventDate<lab2::Julian>,
std::allocator<lab2::EventDate<lab2::Julian> > > lab2::Calendar<lab2::Julian>::eventlist' is
private
calendar.h:166:42: error: within this context
calendar.h: In member function 'lab2::Calendar<T>& lab2::Calendar<T>::operator=(const
lab2::Calendar<T>&) [with T = lab2::Gregorian]':
cprog09lab23.cpp:75:6:   instantiated from 'void {anonymous}::CalendarTester<T, S>::handle() [with
T = lab2::Gregorian, S = lab2::Julian]'
cprog09lab23.cpp:175:13:   instantiated from here
calendar.h:140:5: error: 'const class lab2::Calendar<lab2::Gregorian>' has no member named
'get_eventlist'
calendar.h: In member function 'lab2::Calendar<T>& lab2::Calendar<T>::operator=(const
lab2::Calendar<U>&) [with U = lab2::Julian, T = lab2::Gregorian]':
cprog09lab23.cpp:79:6:   instantiated from 'void {anonymous}::CalendarTester<T, S>::handle() [with
T = lab2::Gregorian, S = lab2::Julian]'
cprog09lab23.cpp:175:13:   instantiated from here
calendar.h:186:5: error: 'const class lab2::Calendar<lab2::Julian>' has no member named
'get_eventlist'
calendar.h:83:29: error: 'std::list<lab2::EventDate<lab2::Julian>,
std::allocator<lab2::EventDate<lab2::Julian> > > lab2::Calendar<lab2::Julian>::eventlist' is
private
calendar.h:190:42: error: within this context
calendar.h:83:29: error: 'std::list<lab2::EventDate<lab2::Julian>,
std::allocator<lab2::EventDate<lab2::Julian> > > lab2::Calendar<lab2::Julian>::eventlist' is
private
calendar.h:190:42: error: within this context
calendar.h: In copy constructor 'lab2::Calendar<T>::Calendar(const lab2::Calendar<T>&) [with T =
lab2::Julian]':
cprog09lab23.cpp:68:6:   instantiated from 'void {anonymous}::CalendarTester<T, S>::handle() [with
T = lab2::Julian, S = lab2::Gregorian]'
cprog09lab23.cpp:177:13:   instantiated from here
calendar.h:98:5: error: 'const class lab2::Calendar<lab2::Julian>' has no member named
'get_eventlist'
```

problem with : circular Dependency between two classes

stack overflow

solution : forward declaration (wiki)

```
red-19:~/c++prog/labbar/laboration2>make datetest.out
g++ -g -std=c++0x -Wall datetest.cpp -o datetest.out

/tmp/ccThhMQs.o: In function `main':
/afs/nada.kth.se/home/e/u1woj9ie/c++prog/labbar/laboration2/datetest.cpp:25: undefined reference
to `set_k_time(long)'
/afs/nada.kth.se/home/e/u1woj9ie/c++prog/labbar/laboration2/datetest.cpp:29: undefined reference
to `lab2::Julian::Julian()'
/afs/nada.kth.se/home/e/u1woj9ie/c++prog/labbar/laboration2/datetest.cpp:30: undefined reference
to `lab2::Gregorian::Gregorian()'
/afs/nada.kth.se/home/e/u1woj9ie/c++prog/labbar/laboration2/datetest.cpp:31: undefined reference
to `lab2::operator<<(std::basic_ostream<char, std::char_traits<char> >&, lab2::Date const&)'
/afs/nada.kth.se/home/e/u1woj9ie/c++prog/labbar/laboration2/datetest.cpp:32: undefined reference
to `lab2::operator-(lab2::Date const&, lab2::Date const&)'
/afs/nada.kth.se/home/e/u1woj9ie/c++prog/labbar/laboration2/datetest.cpp:33: undefined reference
to `lab2::Gregorian::Gregorian(int, int, int)'
/afs/nada.kth.se/home/e/u1woj9ie/c++prog/labbar/laboration2/datetest.cpp:41: undefined reference
to `lab2::Julian::Julian()'
/afs/nada.kth.se/home/e/u1woj9ie/c++prog/labbar/laboration2/datetest.cpp:43: undefined reference
to `lab2::Julian::Julian(lab2::Date const&)'
/afs/nada.kth.se/home/e/u1woj9ie/c++prog/labbar/laboration2/datetest.cpp:45: undefined reference
to `lab2::Julian::Julian(lab2::Date const*)'
/afs/nada.kth.se/home/e/u1woj9ie/c++prog/labbar/laboration2/datetest.cpp:76: undefined reference
to `lab2::Date::operator+=(int)'
/afs/nada.kth.se/home/e/u1woj9ie/c++prog/labbar/laboration2/datetest.cpp:78: undefined reference
to `lab2::Date::operator-=(int)'
/afs/nada.kth.se/home/e/u1woj9ie/c++prog/labbar/laboration2/datetest.cpp:85: undefined reference
to `lab2::Gregorian::Gregorian()'
/afs/nada.kth.se/home/e/u1woj9ie/c++prog/labbar/laboration2/datetest.cpp:86: undefined reference
to `lab2::Gregorian::year() const'
/afs/nada.kth.se/home/e/u1woj9ie/c++prog/labbar/laboration2/datetest.cpp:88: undefined reference
to `lab2::operator==(lab2::Date const&, lab2::Date const&)'
/afs/nada.kth.se/home/e/u1woj9ie/c++prog/labbar/laboration2/datetest.cpp:89: undefined reference
to `lab2::operator!=(lab2::Date const&, lab2::Date const&)'
/afs/nada.kth.se/home/e/u1woj9ie/c++prog/labbar/laboration2/datetest.cpp:90: undefined reference
to `lab2::operator<(lab2::Date const&, lab2::Date const&)'
/afs/nada.kth.se/home/e/u1woj9ie/c++prog/labbar/laboration2/datetest.cpp:91: undefined reference
to `lab2::operator>=(lab2::Date const&, lab2::Date const&)'
/afs/nada.kth.se/home/e/u1woj9ie/c++prog/labbar/laboration2/datetest.cpp:95: undefined reference
to `lab2::Gregorian::Gregorian(int, int, int)'
/afs/nada.kth.se/home/e/u1woj9ie/c++prog/labbar/laboration2/datetest.cpp:127: undefined reference
to `lab2::Gregorian::Gregorian(int, int, int)'
/tmp/ccThhMQs.o: In function `~Roman':
/afs/nada.kth.se/home/e/u1woj9ie/c++prog/labbar/laboration2/roman.h:63: undefined reference to
`vtable for lab2::Roman'
/tmp/ccThhMQs.o: In function `~Julian':
/afs/nada.kth.se/home/e/u1woj9ie/c++prog/labbar/laboration2/julian.h:37: undefined reference to
`vtable for lab2::Julian'
/tmp/ccThhMQs.o: In function `~Gregorian':
/afs/nada.kth.se/home/e/u1woj9ie/c++prog/labbar/laboration2/gregorian.h:36: undefined reference
to `vtable for lab2::Gregorian'
/tmp/ccThhMQs.o: In function `Roman':
```

/afs/nada.kth.se/home/e/u1woj9ie/c++prog/labbar/laboration2/roman.h:35: undefined reference to `vtable for lab2::Roman'
/tmp/ccThhMQs.o: In function `Julian':
/afs/nada.kth.se/home/e/u1woj9ie/c++prog/labbar/laboration2/julian.h:37: undefined reference to `vtable for lab2::Julian'
collect2: ld returned 1 exit status
make: *** [datetest.out] Error 1

g++ -c -std=c++0x -Wall file.cpp -o file.out. for every file.

then

g++ -g -std=c++0x -Wall file1.out file2.out file3.out ...

\ datetest.out → permission denied.

g++ main.cpp definitions.cpp -o main.out

- mer Begränsningar
- Ut. student 2006 KTH
- Quot, bulk

A	&	B	V
V		V	V
F		V	F
V		F	F
F		F	F

- L12635 : D20 ; 2007-09-03

```
214  }
215
216  template <class T>
217  bool Calendar<T>::remove_event(std::string ev_name, int day, int month) {
218      return remove_event(ev_name, day, month, active_date->year());
219  }
220
221  template <class T>
222  bool Calendar<T>::remove_event(std::string ev_name, int day, int month, int year)
223  {
224          try {
225                  T dummy(year,month,day);
226          } catch (std::out_of_range e) { return false; }
227
228          T tDate(year, month, day);
229
230          if (!the_list.empty()) {
231                  typename std::list<cal_linked_list<T>>::iterator it;
232                  for (it=the_list.begin(); it!=the_list.end(); ++it)
233                          if (*it->date == tDate && it->event == ev_name) { the_list.erase(it) ;
    return true; }
234          }
235          return false;
236  }
237
238  template <typename T>
239  ymd Calendar<T>::get_date() const {
240          ymd tDate;
241          tDate.month=active_date->month();
242          tDate.day=active_date->day();
243          tDate.year=active_date->year();
244          return tDate;
245  }
246
247  template <class T>
248  std::ostream& Calendar<T>::print_yourself(std::ostream& os) const{
249
250          if (!the_list.empty()) {
251                  typename std::list<cal_linked_list<T>>::const_iterator it;
252                  for (it=the_list.begin(); it!=the_list.end(); ++it) {
253                          if ( *it->date > *active_date) //efter aktuellt datum
254                                  os << *it->date << " : " << it->event << "\n";
255                  }
256          }
257
258          return os;
259  }
260
261
262
263  template <class T>
264  std::ostream & operator<<(std::ostream & os,const Calendar<T>& c){
265      c.print_yourself(os);
266      return os;
267  }
268
269  }
270  #endif
```

```
144          delete active_date;
145          clean_up();
146
147          ymd tDate= other.get_date();
148          S mDate(tDate.year,tDate.month,tDate.day);
149          active_date = new T(mDate);
150
151          if (!other.get_the_list().empty()) {
152                  typename std::list<cal_linked_list<S>>::const_iterator it;
153                  for (it=other.get_list_begin(); it!=other.get_list_end(); it++) {
154                          T ourDate(*it->date);           // cast it to T calendar dates
155                          add_event(it->event, ourDate.day(),ourDate.month(),ourDate.year());
156                  }
157          }
158          return *this;
159  }
160
161  template <typename T>
162  bool Calendar<T>::set_date(int year, int month, int day) {
163          try {
164                  T dummy(year,month,day);
165          } catch (std::out_of_range e) { return false; }
166
167          delete active_date;
168          active_date=new T(year,month,day);
169          return true;
170  }
171
172  template <class T>
173  bool Calendar<T>::add_event(std::string ev_name) {
174      return add_event(ev_name,active_date->day(),active_date->month(),active_date->year());
175  }
176
177  template <class T>
178  bool Calendar<T>::add_event(std::string ev_name, int day) {
179      return add_event(ev_name,day,active_date->month(),active_date->year());
180  }
181
182  template <class T>
183  bool Calendar<T>::add_event(std::string ev_name, int day, int month) {
184      return add_event(ev_name,day,month,active_date->year());
185  }
186
187  template <class T>
188  bool Calendar<T>::add_event(std::string ev_name, int day, int month, int year) {
189          try {
190                  T dummy(year,month,day);
191          } catch (std::out_of_range e) { return false; }
192
193          cal_linked_list<T> new_element(ev_name, day, month, year);
194
195          if (!the_list.empty()) {
196                  typename std::list<cal_linked_list<T>>::iterator it;
197                  for (it=the_list.begin(); it!=the_list.end(); ++it) {
198                          if ( *it->date == *new_element.date && it->event == ev_name) return
    false; //finns redan
199                          if ( *it->date > *new_element.date) { the_list.insert(it,new_element);
    return true; }
200                  }
201          }
202          the_list.push_back(new_element);
203          return true;
204  }
205
206  template <class T>
207  bool Calendar<T>::remove_event(std::string ev_name) {
208      return remove_event(ev_name, active_date->day(), active_date->month(), active_date->year());
209  }
210
211  template <class T>
212  bool Calendar<T>::remove_event(std::string ev_name, int day) {
213      return remove_event(ev_name, day, active_date->month(), active_date->year());
```

```
1   #include "julian.h"
2   #include "gregorian.h"
3
4   namespace lab2 {
5
6       template <class T>
7       class cal_linked_list {
8       public:
9           std::string event;
10          T* date;
11
12          cal_linked_list();
13              cal_linked_list(const cal_linked_list& other);
14          cal_linked_list(std::string ev_name, int day, int month, int year);
15              ~cal_linked_list();
16          T& get_date_obj();
17
18      };
19
20  template <class T>
21  cal_linked_list<T>::cal_linked_list() {
22          event="";
23          date=new T();
24  }
25
26  template <class T>
27  cal_linked_list<T>::cal_linked_list(const cal_linked_list& other) {
28          date=new T(other.date);
29          event=other.event;
30  }
31
32  template <class T>
33  cal_linked_list<T>::cal_linked_list(std::string ev_name, int day, int month, int year) {
34          event=ev_name;
35          date=new T(year,month,day);
36  }
37
38  template <class T>
39  cal_linked_list<T>::~cal_linked_list() {
40          delete date;
41  }
42  template <typename T>
43  T& cal_linked_list<T>::get_date_obj() {
44      return *date;
45  }
46  }
```

```
1   #ifndef ___CALENDAR_H___
2   #define ___CALENDAR_H___
3
4   #include "date.h"
5   #include "julian.h"
6   #include "gregorian.h"
7   #include <iostream>
8   #include "stdlib.h"
9   #include <stdexcept>
10  #include "callinked.h"
11  #include <list>
12
13  namespace lab2 {
14
15    template <typename T>
16    class Calendar {
17    public:
18      Calendar();
19      Calendar(int year, int month, int day):the_list()
20        { active_date=new T(year,month,day); }
21
22      Calendar(const Calendar& other);
23      template <typename S> Calendar(const Calendar<S>& other);
24      ~Calendar();
25
26      Calendar<T>& operator=(const Calendar<T>& other);
27
28      template <typename S> Calendar& operator=(const Calendar<S>& other);
29
30      bool set_date(int year, int month, int day);
31      bool add_event(std::string ev_name);
32      bool add_event(std::string ev_name, int day);
33      bool add_event(std::string ev_name, int day, int month);
34      bool add_event(std::string ev_name, int day, int month, int year);
35
36      bool remove_event(std::string ev_name);
37
38      bool remove_event(std::string ev_name, int day);
39
40      bool remove_event(std::string ev_name, int day, int month);
41
42      bool remove_event(std::string ev_name, int day, int month, int year);
43
44      std::list<cal_linked_list<T>> get_the_list() const
45          { return the_list; }
46
47      typename std::list<cal_linked_list<T>>::const_iterator get_list_begin() const
48      { return the_list.begin(); }
49      typename std::list<cal_linked_list<T>>::const_iterator get_list_end() const
50      { return the_list.end(); }
51
52      T* get_active_date() const { return active_date; }
53
54      ymd get_date() const;
55
56      std::ostream& print_yourself(std::ostream& os) const;
57      friend std::ostream & operator<<(std::ostream & os,const Date& d);
58
59    private:
60      void clean_up();
61      std::list<cal_linked_list<T>> the_list;
62      T* active_date;
63
64  };
65
66  //============================================================================
67
68  template <typename T>
69  Calendar<T>::Calendar(): the_list() {
70              active_date=new T();
71        }
72
```

```
 73  template <typename T>
 74  Calendar<T>::Calendar(const Calendar& other): the_list()
 75  {
 76    //copy constructor
 77    ymd tDate= other.get_date();
 78    T mDate(tDate.year,tDate.month,tDate.day);
 79    active_date = new T(mDate);
 80
 81    if (!other.get_the_list().empty()) {
 82    //typename std::list<cal_linked_list<T>>::const_iterator it;
 83      for (it=other.get_list_begin(); it!=other.get_list_end(); it++) {
 84        add_event( it->event, it->date->day(), it->date->month(), it->date->year());
 85      }
 86    }
 87  }
 88
 89  template <typename T>
 90  template <typename S>
 91  Calendar<T>::Calendar(const Calendar<S>& other): the_list()
 92  {
 93    //copy constructor
 94    ymd tDate= other.get_date();
 95    S mDate(tDate.year,tDate.month,tDate.day);
 96    active_date = new T(mDate);
 97
 98    if (!other.get_the_list().empty()) {
 99    //typename std::list<cal_linked_list<S>>::const_iterator it;
100      for (it=other.get_list_begin(); it!=other.get_list_end(); it++) {
101        T ourDate(*it->date);              // cast it to T calendar dates
102        add_event(it->event, ourDate.day(),ourDate.month(),ourDate.year());
103      }
104    }
105  }
106
107  template <typename T>
108  Calendar<T>::~Calendar() {
109          clean_up();
110          delete active_date;
111  }
112
113  template <typename T>
114  void Calendar<T>::clean_up() {
115          the_list.clear();
116  }
117
118  template <typename T>
119  Calendar<T>& Calendar<T>::operator=(const Calendar<T>& other) {
120          //Implement copy procedure
121          if (this==&other) return *this;
122
123          delete active_date;
124          clean_up();
125
126          ymd tDate= other.get_date();
127          T mDate(tDate.year,tDate.month,tDate.day);
128          active_date = new T(mDate);
129
130          if (!other.get_the_list().empty()) {
131                //typename std::list<cal_linked_list<T>>::const_iterator it;
132                for (it=other.get_list_begin(); it!=other.get_list_end(); ++it) {
133                        add_event(it->event , it->date->day(), it->date->month(), it->date->year
    ());
134                }
135          }
136
137          return *this;
138  }
139
140  template <typename T>
141  template <typename S>
142  Calendar<T>& Calendar<T>::operator=(const Calendar<S>& other) {
143          //Implement copy procedure
```

```
#include "julian.h"

namespace lab2 {
//Ctor - default
        Julian::Julian() : roman() {validate_date(year(),month(),day());}

//Ctor for y,m,d
        Julian::Julian(int y, int m, int d) : roman(0) {
                validate_date(y,m,d);
                Date::days_from_zero=julian_to_jd(y,m,d);
        }
//Ctor for Date
        Julian::Julian(const Date& date) :roman(0) {
                Date::days_from_zero=date.get_days_from_zero();
                validate_date(year(),month(),day());
        }
//Ctor for Date*
        Julian::Julian(const Date* date) :roman(0) {
                Date::days_from_zero=date->get_days_from_zero();
                validate_date(year(),month(),day());
        }

        int Julian::year() const
        { return jd_to_julian(Date::days_from_zero).year; }

        int Julian::month() const
        { return jd_to_julian(Date::days_from_zero).month; }

        int Julian::day() const
        { return jd_to_julian(Date::days_from_zero).day; }

        bool Julian::is_leap_year(int y) const
        { return y%4==0; }

        int Julian::days_this_month() const {
            int m=month();
            int i=0;
            for (; i<31; i++) {
                ymd tm=jd_to_julian(Date::days_from_zero+i);
                if (tm.month!=m) break;
            }
            return jd_to_julian(Date::days_from_zero+i-1).day;
        }

        double Julian::ymd_to_jd(int year, int month, int day) const
        { return julian_to_jd(year,month,day); }

        ymd Julian::jd_to_ymd(double JD) const
        { return jd_to_julian(JD); }

        double Julian::julian_to_jd(int year, int month, int day) const {
        // http://quasar.as.utexas.edu/BillInfo/JulianDatesG.html


                if(month<=2){ year--; month+=12; }
```

```
            double E = floor(365.25*(year+4716));
            double F = floor(30.6001*(month+1));
            double JD= day+E+F-1524.5;
            return JD;
    }

    ymd Julian::jd_to_julian(double JD) const {
            ymd tDate;
            double Z = floor(JD+0.5);
            double A=Z; // for jd to julian
            double B = A+1524;
            double C = floor((B-122.1)/365.25);

            double D = floor(365.25*C);
            double E = floor((B-D)/30.6001);
            double F = floor(30.6001*E);
            double G = Z - floor(Z);
            tDate.day= B-D-F+G;
            // (must get number less than or equal to 12)
            tDate.month = ( E-1<=12) ? E-1: E-13;

            tDate.year = (tDate.month<=2) ? C-4715 : C-4716;
            return tDate;
    }

Date& Julian::operator++()
{
    ++Date::days_from_zero;
    return *this;
}

Date& Julian::operator--()
{
    --Date::days_from_zero;
    return *this;
}

Julian Julian::operator++(int)
{
    Julian source(*this);
    ++(*this);
    return source;
}
Julian Julian::operator--(int)
{
    Julian source(*this);
    --(*this);
    return source;
}
```

```
#include "gregorian.h"

namespace lab2 {

   Gregorian::Gregorian(): roman()
   { validate_date(year(),month(),day());}

    Gregorian::Gregorian(int y, int m, int d) : roman(0)
    {
        validate_date(y,m,d);
        Date::days_from_zero= gregorian_to_jd(y, m , d);
        }

    Gregorian::Gregorian(const Date& date) :roman(0) {
            Date::days_from_zero=date.get_days_from_zero();
                        validate_date(year(),month(),day());
    }

    Gregorian::Gregorian(const Date* date) :roman(0) {
            Date::days_from_zero=date->get_days_from_zero();
                        validate_date(year(),month(),day());
    }


    int Gregorian::year() const
    { return jd_to_gregorian(Date::days_from_zero).year; }

    int Gregorian::month() const
    { return jd_to_gregorian(Date::days_from_zero).month; }

    int Gregorian::day() const
    { return jd_to_gregorian(Date::days_from_zero).day; }

    int Gregorian::days_this_month() const { //pretty cpu hungry
        int m=month();
        int i=0;
        for (; i<31; i++) {
            ymd tm=jd_to_gregorian(Date::days_from_zero+i);
            if (tm.month!=m) break;
        }
        return jd_to_gregorian(Date::days_from_zero+i-1).day;
    }

    double Gregorian::ymd_to_jd(int year, int month, int day) const
    { return gregorian_to_jd(year,month,day); }

    ymd Gregorian::jd_to_ymd(double JD) const
    { return jd_to_gregorian(JD); }

    double Gregorian::gregorian_to_jd(int y, int m, int d) const {
        //http://www.hermetic.ch/cal_stud/jdn.htm
            double JD = ( 1461 * ( y + 4800 + ( m - 14 ) / 12 ) ) / 4 +
          ( 367 * ( m - 2 - 12 * ( ( m - 14 ) / 12 ) ) ) / 12 -
          ( 3 * ( ( y + 4900 + ( m - 14 ) / 12 ) / 100 ) ) / 4 +
          d - 32075  ;
            return JD;
```

```
}

ymd Gregorian::jd_to_gregorian(double JD) const {
        ymd tDate;
        JD=floor(JD+0.5);
        int l = JD + 68569;
        int n = ( 4 * l ) / 146097;
        l = l - ( 146097 * n + 3 ) / 4;
        int i = ( 4000 * ( l + 1 ) ) / 1461001;
        l = l - ( 1461 * i ) / 4 + 31;
        int j = ( 80 * l ) / 2447;
        tDate.day = l - ( 2447 * j ) / 80;
        l = j / 11;
        tDate.month = j + 2 - ( 12 * l );
        tDate.year = 100 * ( n - 49 ) + i + l;
        return tDate;
}



bool Gregorian::is_leap_year(int y) const
{
  bool leap_year=false;
  if ( y%4==0 ) leap_year=true; // skott•r var 4:e •r
  // •rhundraden mod 400 m•ste ox• vara 0 annars ej skott•r
  if ( y%100==0 && y%400!=0) leap_year=false;
        return leap_year;
}

Date& Gregorian::operator++()
{
    ++Date::days_from_zero;
    return *this;
}

Date& Gregorian::operator--()
{
    --Date::days_from_zero;
    return *this;
}

Gregorian Gregorian::operator++(int)
{
    Gregorian source(*this);
    ++(*this);
    return source;

}
Gregorian Gregorian::operator--(int)
{
    Gregorian source(*this);
    --(*this);
    return source;
}
```

```
#include <stdexcept>
#include "roman.h"
#include <math.h>

namespace lab2 {

    int roman::days_per_week() const { return 7; }

    int roman::months_per_year() const { return 12; }

    int roman::week_day() const {
      return ((int)floor(Date::days_from_zero+0.5) % days_per_week()) + 1;// 1
to 7
    }

    std::string roman::week_day_name() const {
            return week_day_name_s[ (int)floor(Date::days_from_zero+0.5)%7  ];
    }

    std::string roman::month_name() const {
            return month_name_s[month()-1];
    }
```

```
    bool roman::add_month(int n) {
        int s=0;
        if (n<0) s=-1; else s=1;
        while (n!=0) {
            if (n>0) {
                int old_day = day(); // This is what we want in the new month
                //int days_left_in_month = days_this_month() - day();
                int days_left_in_month = days_of_month(year(),month()) - day();
                Date::days_from_zero += days_left_in_month+1; // go to first in
 the next month

                ->if(old_day <= days_this_month())
                    { Date::days_from_zero += old_day -1; } // reduce 1 since w
e added 1 earlier
                    ->else { Date::days_from_zero+=30-(days_left_in_month+1); } /
/ we go for 30 days

            } else {
                int old_day = day(); // Backwards in time
                Date::days_from_zero -= old_day; // Go to last day in prev month
h
                //if(old_day <= days_this_month()
)
                if(old_day <= days_of_month(year(),month()))
                    { Date::days_from_zero -= days_this_month() - old_day; }
                    else { Date::days_from_zero-=(30-old_day); }

            }
            n-=s; // reduce month counter
        }
        return true;
    }
```

```
   bool roman::add_year(int n) {
       int y = year();
       int m = month();
       int d = day();
       if (m==2 && d==29 && is_leap_year(y+n)==false) {
           //From leap year to not leap year
```

```
            Date::days_from_zero=ymd_to_jd(y+n,m,28);
        } else {
            //From whatever to whatever
            Date::days_from_zero=ymd_to_jd(y+n,m,d);
        }
        return true;
    }

    int roman::mod_julian_day() const {
        int MJD=floor(Date::days_from_zero - 2400000.5);
        return MJD;
    }

    void roman::validate_date(int year, int month, int day) const {
        //we dont care bout year
        if (month<1 || month>12) throw std::out_of_range("Month out of range");

        if (day<1 || day>days_of_month(year,month)) throw std::out_of_range("Da
y out of range");


    }

    int roman::days_of_month(int year, int month) const {
        switch (month) {
            case 1:
                return 31;
                break;
            case 2:
                if (is_leap_year(year)==true) return 29;
                return 28;
                break;
            case 3:
                return 31;
                break;
            case 4:
                return 30;
                break;
            case 5:
                return 31;
                break;
            case 6:
                return 30;
                break;
            case 7:
                return 31;
                break;
            case 8:
                return 31;
                break;
            case 9:
                return 30;
                break;
            case 10:
                return 31;
                break;
            case 11:
```

```
                return 30;
                break;
            case 12:
                return 31;
                break;
        }
        return 0;
    }
    int roman::days_one_month(int year,int month) const {
        int JD=ymd_to_jd(year,month,1);
        int i=0;
        for (; i<31; i++) {
            ymd tm=jd_to_ymd(JD+i);
            if (tm.month!=month) break;
        }
        return jd_to_ymd(JD+i-1).day;
    }

    const std::string roman::week_day_name_s[]= {"monday", "tuesday", "wednesda
y", "thursday", "friday", "saturday", "sunday"};
    const std::string roman::month_name_s[]   = {"january", "february", "mar
, "april", "may", "june", "july", "august", "september", "october","november","
december"};
}
```