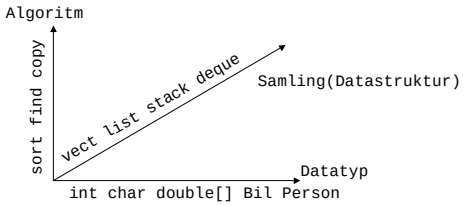
Algoritmbiblioteket (STL)

- (f.d.) Standard Template Library
- Samlingstyper (containers)
- Algoritmer
- Funktionsobjekt
- Adaptrar

Designstrategi

- Generiska algoritmer som fungerar på godtyckliga samlingsdatatyper, vilka har iteratorer



forts designstrategi

- Effektiva algoritmer
- Använder enbart mallar i C++
  - Inga arvshierarkier eller virtuella funktioner
- Ej objekt-orienterat
- Bygger på forskning om generiska algoritmer av Stepanov/Lee/Musser -79-95
  - General Electric, AT&T Bell, HP
- Presenterades för ANSI/ISO C++ 93
  - Mycket gott mottagande

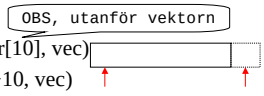
Samlingstyper

- Sekvenser
  - vector<T>,deque<T>,list<T>
- Associativa
  - set<T,C>, map<K,V,C>, multiset<T,C>, multimap<K,V,C>
- Adaptrar
  - stack<T>, queue<T>, priority_queue<T,C>
  - impl t.ex. som vector, deque, eller list

Anm. C comparator. Funktionsobjekt för sortering

Iteratorer

- Pekare
  - vector<int>::iterator it=vec.begin();
  - *it++ = 17;
- Intervallet är [begin,end) dvs begin ingår men ej end
- Vanliga pekare kan användas
  - int arr[10];
  - copy(&arr[0], &arr[10], vec);
  - eller copy(arr, arr+10, vec)



Algoritmer

- Ca 70 st
- Opererar på intervall
  - Oberoende av datastruktur
- Sekvensalgoritmer
  - for_each, find, copy, transform, replace
- Sortering
  - sort, stable_sort, binary_search, max_element
- Numeriska
  - accumulate, inner_product

Funktionsobjekt

- Objekt med funktionsanropsoperator
  - T::operator()
- Bättre än funktionspekare
  - Bindning vid kompileringen
  - Kan innehålla data
- ”Modern” programmering

```
set<int, less<int> > s;
remove_if(first,last,not1(bind2nd(modulus<int>(),2)));
```

”ta bort alla jämna tal”

Ett litet exempel

```
void sort_file(char *name)
{
  ifstream file(name);
  istream_iterator<string,ptrdiff_t> in(file),eof;
  vector<string> strs;

  copy(in,eof,back_inserter(strs));
  sort(strs.begin(),strs.end());
  copy (strs.begin(),strs.end(),
    ostream_iterator<string>(cout,”\n”));
}
```


Användning (forts exempel)

```
void main(int argc, char *argv[])
{
  for_each(argv+1,argv+argc,sort_file);
}
```


Vad är en iterator

- Pekarabstraktion
- Läsiterator
  - för att avläsa värden
- Skriviterator
  - enbart för tilldelning
- Enkel/dubbelriktad
  - stegar i en/båda riktning/-arna läs&skriv
- Löper över en samling element, en i taget
- Ofta en-gångs objekt

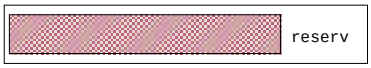
Iteratorexempel

```
list<int> lst(10);
list<int>::iterator p=lst.begin();
while (p != lst.end())
{
 *p = 37;
 p++;
};
```


Samlingstyper, vector<T>

- index från 0
- range-Check med .at()
- minneshantering&storlek automatiskt

```
#include <vector>
using namespace std;
vector<int> vec(10);
vec[0]=12; vec[1]=37;
vec.push_back(1234); // vec[10] = 1234;
```



2

Medlemsfunktioner

- Konstruktor
  - vector(), vector(int size), vector(int size, T value)
- Aktuellt antal element
  - vec.size(); vec.empty();
- Kapacitet
  - vec.capacity();
- Indexering
  - vec[i], vec.at(i), vec.front(), vec.back()
- Iterator
  - vec.begin(), vec.end()

list<T>

- Dubbellänkad, dynamisk lista
- Insättning och uttag via iterator O(n)
- Access av ändarna O(1)
- Saknar indexering

```
#include <list>
list<int> lst;
lst.push_front(123); lst.push_back(987);
```


deque<T>

- double-ended queue ”deck”
- mellanting lista & vektor
- använder minnesblock

Mängd set<T,Comparator>

- Ordnad mängd `ofta impl som binära träd`
  - set - unika element
  - multiset - ev flera lika

```
   Comparator function object, anropar operator<
#include <set>        Obs, mellanslag!
set<int, less<int> > s;
s.insert(12); s.insert(34);
set_union(s.begin(),s.end(),s2.begin(),s2.end(),
        inserter(sr,sr.begin()));
```


Utskriftsoperator för mängd

```
#include <set>

using namespace std;

typedef set<double,less<double> > set_type;

ostream& operator<<(ostream& out, const set_type& s)
{
  copy(s.begin(), s.end(),
    ostream_iterator<set_type::value_type>(out," "));
  return out;
}
```


map<Key,T,Comparator>

- Associativ, ordnad vektor
  - map unika nycklar
  - multimap flera lika nycklar

```
#include <map>
map<string, int, less<string> > dayInMonth;
dayInMonth[”December”] = 31;
```

Krav på elementen T

- Defaultkonstruktor T::T()
- Kopieringskonstruktor T::T(const T& t)
- Tilldelningsoperator
  - T& T::operator=(const T& t)
- Destruktor T::~T()
- Likhet x == y
- Ordning (för set/map) x<y
- ”Ortodox kanonisk klassform”

Algoritmer

- Generiska mha funktionsmallar
- Arbetar på intervall
- Fungerar även på standardtyper int a[5];

Icke-destruktiva sekvensalgoritmer

- for_each
  - applic en fkn på varje element
- find
  - letar; pos = find(lst.begin(),lst.end(),”ord”);
  - cout << *pos;
  - find_if(InputIteratorBegin,InputIteratorEnd, UnaryPredicate)

Exempel på find_if

```
bool div_by_ten(int n) { return !(n%10);}

void main()
{
  vector<int> vec(100);
  vector<int>::iterator pos;
  pos = find_if(vec.begin(),
            vec.end(),div_by_ten);
  cout << *pos;
}
```


Fler algoritmer...

- count
  - void count(InputIteratorBegin,InputIteratorEnd, Value, size& n)
  - count_if
- equal / mismatch
  - Ser om två samlingar är lika
- search
  - letar efter delintervall

Destruktiva sekvensalgoritmer

- copy
- remove
  - tar bort element med viss egenskap
- replace
- fill
- generate
  - fyller med generatorfunktion
- transform
  - applicerar funktion på varje element

Sorteringsalgoritmer

- sort O(n log n) worst:O(n2)
- binary_search
- min_element,max_element

Mängdrelaterade alg

- includes
- set_union, set_intersection
- accumulate
- transform
- inner_product

Funktionsobjekt

- Generalisering av funktionspekare
- Objekt med funktionsanropsoperator
  - T funcObj::operator()(const T&)

```
class LinTrans
{ public:
  LinTrans(double a, double b): A(a), B(b) {}
double operator()(double x) { return A*x+B; }
private: double A,B; };
Lintrans t(2.34,5.67);
double y=t(0.45);
```


Funktionstyper i STL

- plus<T>, minus<T>, times<T>, divides<T>, modulus<T>, negate<T>
- equal<T>, less<T>, greater<T>, …
  - sort(v.begin(),v.end(),greater<string>());//stig ordn
- logical_and<T>, ..

Argumentbindare

- Gör om binär fkn till unär genom att binda ett av argumenten
  - bind1st, bind2nd

```
vector <double> v;
int mean = accumulate(v.begin(), v.end(), 0,
      plus<double>()) / v.size();
transform(v.begin(),v.end(),
      bind2nd(minus<double>(), mean));
```