String in C

nadelen

- statisch (lengte tijdens compileren bepaald).
- null karakter als afsluiting.
- operatoren zijn niet bruikbaar (de str*xxx* functies moeten gebruikt worden).

```
#include <string.h>
#include <iostream.h>
int main() {
   char naam[10];
//  naam="Harry"; // Error!
   strcpy(naam, "Harry"); // OK
   cout<<naam<<endl;
   strcpy(naam, "Willem-Alexander"); // Error!
   cout<<naam<<endl;
//  if (naam=="Kees") { // error
   if (strcmp(naam, "Kees")==0) { // OK
...
```

THRijswijk

string in C++

voordelen

- dynamisch (lengte tijdens run-time bepaald).
- operatoren zijn bruikbaar.
- veel extra functionaliteit.

```
#include <string>
#include <iostream>
using namespace std;

int main() {
   string naam;
   naam="Harry";
   cout<<naam<<endl;
   naam="Willem-Alexander";
   cout<<naam<<endl;
   if (naam=="Kees") {
      cout<<"Niet goed!"<<endl;
   }
   return 0;
}
```

THRijswijk

string-bibliotheek

- vergelijken met operatoren **> < >= <= == !=**
- optellen en toevoegen met operatoren **+** en **+=**
- **insert** voor invoegen
- **erase** voor verwijderen
- **replace** voor vervangen
- **find** voor zoeken
- **c_str()** voor conversie naar const char*

```
string s;
cout<<"Geef filenaam: “;
cin>>s;
ifstream fin(s.c_str());
getline(fin, s);
```

THRijswijk

string

Iterator

Met behulp van een **iterator** kun je door een string lopen op dezelfde manier als dat je met een **pointer** door een char[] kunt lopen.

```
char naam1[ ]="Henk";
for (const char* p(naam1); *p!='\0'; ++p)
   cout<<*p<<" ";
cout<<endl;
cout<<"De eerste letter is:"
      <<naam1[0]<<endl;
cout<<"De laatste letter is: "
      <<naam1[strlen(naam1)-1]<<endl;

string naam2("Harry");
for (string::const_iterator i(naam2.begin());
      i!=naam2.end(); ++i)
   cout<<*i<<" ";
cout<<endl;
cout<<"De eerste letter is:"
      <<naam2[0]<<endl;
cout<<"De laatste letter is: "
      <<naam2[naam2.size()-1]<<endl;
```

THRijswijk

STL

- Containers (datastructuren)
- Iterators (om door een container te wandelen)
- Algoritmen (generiek)

Het gebruik van iterators maakt generiek programmeren (**generic programming**) mogelijk. Een generiek algoritme kan op verschillende datatypes (containers) toegepast worden.

Generic programming is dus niet OOP.

Met behulp van iterators kun je een algoritme op de objecten in een container uitvoeren.

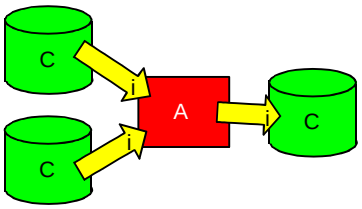
De algoritmen zijn zo efficient mogelijk (geen inheritance en virtuele functies gebruikt).

Je kunt eigen algoritmen, containers en iteratoren toevoegen.

THRijswijk

STL

componenten



C = container
A = algoritme
i = iterator

Een algoritme kan ook uit 1 container lezen en in dezelfde container schrijven

THRijswijk

Containers

- Sequentiële containers (linear)
  - []
  - **vector**
  - **list**
  - **deque** (**d**ouble **e**nded **que**ue eigenlijk een devector)
  - bitset (slaan wij over)
  - adapters (in combinatie met vector, list of deque)
    - **stack**
    - **queue**
    - **priority_queue**
- Associatieve containers (elementen op volgorde van sleutelwaarde opgeslagen)
  - **set** (key orgineel)
  - **multiset** (key meerdere malen)
  - **map** (key, value orgineel)
  - **multimap** (key, value meerdere malen)

THRijswijk

Containers

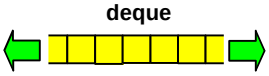
- De containers regelen hun eigen **geheugenbeheer**.
- De containers maken **kopietjes** van de elementen die erin gestopt worden. Als je dat niet wilt dan moet je een container met pointers gebruiken.
- Alle objecten in een container zijn van **het zelfde type**. (Dat kan wel een polymorph pointertype zijn!)

THRijswijk

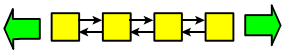
Sequentiële containers

vector

deque



list



THRijswijk

Sequentiële containers

- **Vector**
  - random access
    - operator[ ]
    - at(...) gooit exception out_of_range
  - toevoegen of verwijderen
    - push_back(...) pop_back() O(1)
    - insert(...) erase(...) O(N)
- **Deque**
  - random access zoals vector
  - toevoegen of verwijderen
    - push_back(...) pop_back() O(1)
    - push_front(...) pop_front() O(1)
    - insert(...) erase(...) O(N)
- **List**
  - **geen** random access
  - toevoegen of verwijderen is O(1)
    - push_back(...) pop_back() O(1)
    - push_front(...) pop_front() O(1)
    - insert(...) erase(...) **O(1)**

THRijswijk

Vector voorbeeld

```
#include <iostream>
#include <vector>
using namespace std;

int main() {
   vector<int> v;
   int i;
   cin>>i;
   while (i!=0) {
      v.push_back(i);
      cin>>i;
   }
   print(v); // zelf gedefinieerde functie print
   // Een deel bewerken met iterator.
   if (v.size()>=4)
      for (vector<int>::iterator iter(v.begin()+2);
            iter!=v.begin()+4; ++iter)
         *iter*=2;
   print(v);
   // Een deel bewerken met index.
   if (v.size()>=4)
      for (vector<int>::size_type i(2); i<4; ++i)
         v[i]/=2;
   print(v);
...
```

THRijswijk

Vector voorbeeld

```
// print met behulp van een index:

void print(vector<int>& vec) {
   cout<<"De inhoud van de vector is:"<<endl;
   for (vector<int>::size_type
         index(0); index!=vec.size(); ++index) {
      cout<<vec[index]<<" ";
   }
   cout<<endl;
}

// print met behulp van een iterator:

void print(vector<int>& vec) {
   cout<<"De inhoud van de vector is:"<<endl;
   for (vector<int>::const_iterator
         iter(vec.begin()); iter!=vec.end(); ++iter) {
      cout<<*iter<<" ";
   }
   cout<<endl;
}
```

THRijswijk

List voorbeeld

```
#include <iostream>
#include <list>
using namespace std;

class Hond {
public:
   virtual void blaf() const =0;
...
};
class Tekkel: public Hond {
...
}
class SintBernard: public Hond {
...
}

int main() {
   list<Hond*> kennel;
   kennel.push_back(new Tekkel);
   kennel.push_back(new SintBernard);
   kennel.push_back(new Tekkel);
   for (list<Hond*>::const_iterator
         i(kennel.begin(); i!=kennel.end(); ++i)
      (*i)->blaf();
...
```

THRijswijk

Adapters

- **stack**
  - push(...), pop(), top(), empty()
- **queue**
  - kan niet met vector.
  - push(...), pop(), front(), empty()
    Let op: push voegt achteraan toe en pop verwijderd vooraan.
- **priority_queue**
  - kan niet met list.
  - push(...), pop(), top(), empty()
    Let op: top geeft het grootste element terug en pop verwijderd dit element.
  - wordt geïmplementeerd als binary heap.

Deze adapters passen de interface van een vector, deque of list aan.

```
stack<int, vector<int> > s1;
stack<int, deque<int> > s2;
stack<int, list<int> > s3;
stack<int> s4; // using deque by default
```

THRijswijk

Stack voorbeeld

```
#include <iostream>
#include <stack>
using namespace std;

int main() {
   stack<char> s;
   char c; cin.get(c);
   while (c!='.') {
      if (c=='('||c=='{'||c=='[')
         s.push(c);
      else
         if (c==')'||c=='}'||c==']')
            if (s.empty())
               cout<<"Fout"<<endl;
            else {
               char d(s.top());
               s.pop();
               if (d=='('&&c!=')'||
                     d=='{'&&c!='}'||d=='['&&c!=']')
                  cout<<"Fout"<<endl;
            }
      cin.get(c);
   }
   if (!s.empty())
      cout<<"Fout"<<endl;
...
```

THRijswijk

Associatieve containers

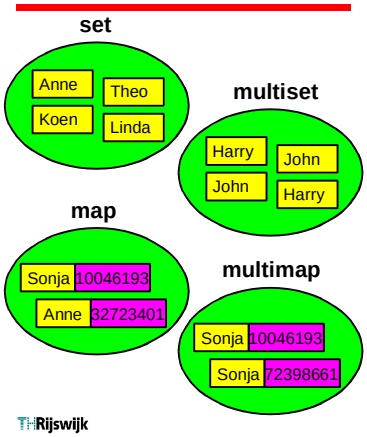
- **set** (verzameling)
- **multiset** (bag)
- **map** (1:1 relatie)
- **multimap** (1:N relatie)

Mogelijkheden
- zoeken op key
- doorlopen op volgorde van key

Implementatie
- hash table
- binary search tree (in std gebruikt)

THRijswijk

Associatieve containers



THRijswijk

Set (verzameling)

- Toevoegen met **insert**
  - Een element wordt automatisch op de "goede" plaats ingevoegd.
  - Returntype is een pair<iterator, bool>. De bool geeft aan of het invoegen gelukt is en de iterator geeft de plek aan waar ingevoegd is.
- Verwijderen met **erase**
  - Een te verwijderen element wordt automatisch opgezocht.
- Zoeken met **find**
  - Geeft een iterator terug naar de plaats waar het element staat en geeft end() terug als element niet gevonden is.
- Zoeken met **count**
  - Geeft het aantal keer dat een element voorkomt (0 of 1).

THRijswijk

Set voorbeeld

```
#include <iostream>
#include <string>
#include <set>
using namespace std;

void print(set<string>& s) {
   cout<<"De set bevat: ";
   for (set<string>::iterator i(s.begin());
        i!=s.end(); ++i)
      cout<<*i<<" ";
   cout<<endl;
}

int main() {
   set<string> docenten;
   docenten.insert("John");
   docenten.insert("Paul");
   docenten.insert("Paul");
   docenten.insert("Harry");
   docenten.insert("Bart"); print(docenten);
   pair<set<string>::iterator, bool>
      result(docenten.insert("Harry"));
   if (result.second==false)
      cout<<"1 Harry is genoeg."<<endl;
   docenten.erase("Harry"); print(docenten);
...
```

THRijswijk

MultiSet (bag)

- Toevoegen met **insert**
  - Een element wordt automatisch op de "goede" plaats ingevoegd.
  - Returntype is een iterator. Deze iterator geeft de plek aan waar ingevoegd is.
- Verwijderen met **erase**
  - Een te verwijderen element wordt automatisch opgezocht. Alle gevonden elementen worden verwijderd.
- Zoeken met **find**
  - Geeft een iterator terug naar de plaats waar het eerste gevonden element staat en geeft end() terug als element niet gevonden is.
- Zoeken met **count**
  - Geeft het aantal keer dat een element voorkomt (>=0).

THRijswijk

Map

1:1 relatie

- Elementen zijn:
  pair<const key, value>
- interface gelijk aan set
  - insert
  - erase
  - find
  - count
- extra **operator[ ]**
  - met operator[ ] kun je een key als index gebruiken.
  - Als de key al in de map zit wordt een reference naar de bijbehorende value teruggegeven.
  - Als de key niet aanwezig in de map dan wordt deze key toegevoegd met de default value (default constructor).

THRijswijk

Map voorbeeld

```
#include <iostream>
#include <fstream>
#include <string>
#include <map>
using namespace std;

int main() {
   string w;
   map<string, int> freq;
   cout<<"Geef filenaam: ";
   cin>>w;
   ifstream fin(w.c_str());
   while (fin>>w) {
      ++freq[w];
   }
   for (map<string, int>::const_iterator
        i(freq.begin()); i!=freq.end(); ++i) {
      cout<<i->first<<" "<<i->second<<endl;
   }
   cout<<"Enkele keywords:"<<endl;
   cout<<"if: "<<freq["if"]<<endl;
   cout<<"while: "<<freq["while"]<<endl;
...
```

THRijswijk

Multimap

1:N relatie

- Elementen zijn:
  pair<const key, value>
- interface gelijk aan multiset
  - insert
  - erase
  - find
  - count
- **geen** operator[ ]

THRijswijk

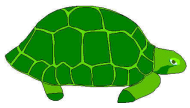
Iterators

Een iterator is geen class maar een (interface) afspraak. Elke class die aan deze afspraak voldoet is als iterator te gebruiken.

- **input** iterator (single pass)
  - een voor een lezen * (rvalue) ++
- **output** iterator (single pass)
  - een voor een schrijven * (lvalue) ++
- **forward** iterator
  - voorwaarts * ++ == !=
- **bidirectional** iterator
  - voor-/achterwaarts * ++ --== !=
- **random access** iterator
  - met sprongen * ++ -- += -= + -== != > >= < <= [ ]
  - een gewone pointer is een random access iterator

THRijswijk

Iteratoren



Forward iterator

Bidirectional iterator

Random access iterator

THRijswijk

Iterators

Relatie met containers

- bidirectional
  - list, set, multiset, map en multimap
- random access
  - vector en deque

Iterators

Relatie met algoritmes

- input bijvoorbeeld find
- output bijvoorbeeld copy
- forward bijvoorbeeld remove
- bidirectional bijvoorbeeld reverse
- random access bijvoorbeeld sort

THRijswijk

Iterators

Speciale iterators

- Insert-iterators (een soort cursor in insert mode).
  - **insert_iterator**
  - **back_insert_iterator**
  - **front_insert_iterator**
- Stream-iterators (koppeling tussen iostream lib en STL).
  - **istream_iterator**
  - **ostream_iterator**

```
vector<int> rij;
ifstream fin("getallen.txt");
istream_iterator<int> iin(fin);
istream_iterator<int> einde;
copy(iin, einde, back_inserter(rij));
sort(rij.begin(), rij.end());
ostream_iterator<int> iout(cout, " ");
copy(rij.begin(), rij.end(), iout);
```

THRijswijk

Algoritmen

Een algoritme werkt op een **sequence**. Een sequence wordt aangegeven door twee iteratoren i1 en i2. De sequence loopt van i1 **tot** (dus **niet** t/m) i2.

- sequence operations
  - non-mutating
  - mutating
- sorteren en aanverwanten
- functie-objecten

```
vector<int> v;
v.push_back(12);
v.push_back(18);
v.push_back(6);
sort(v.begin(), v.end());
for (vector<int>::size_type i(0); i<v.size(); ++i)
   cout<<v[i]<<" ";
```

THRijswijk

Algoritmen

Non-mutating

- **Zoeken** van elementen
  - find
  - find_if
  - find_first_of
- **Tellen** van elementen
  - count
  - count_if
- **Werken** met elementen
  - for_each
- **Zoeken** naar een sequence
  - search
  - find_end
  - adjacent_find
  - search_n
- **Vergelijken** van sequences
  - mismatch
  - equal

THRijswijk

find

```
#include <string>
#include <set>
#include <iostream>
#include <algorithm>
using namespace std;

int main() {
   string s("galgje");
   set<int> v;
   do {
      for (string::size_type i(0); i<s.size(); ++i)
         if (find(v.begin(), v.end(), i)==v.end())
            cout<<'.';
         else
            cout<<s[i];
      cout<<endl<<"Geef een letter: ";
      char c;
      cin>>c;
      string::iterator r(s.begin());
      while ((r=find(r, s.end(), c))!=s.end()) {
         v.insert(r-s.begin());
         ++r;
      }
   }
   while (v.size()!=s.size());
   cout<<"Je hebt het woord gevonden:"<<s;
```

THRijswijk

find_if

```
#include <iostream>
#include <list>
#include <algorithm>
using namespace std;

bool ispos(int i) {
   return i>=0;
}

int main() {
   list<int> l;
   l.push_back(-3);
   l.push_back(-4);
   l.push_back(3);
   l.push_back(4);
   list<int>::iterator r;
   r=find_if(l.begin(), l.end(), ispos);
   if (r!=l.end())
      cout<<"Het eerste positieve element is:"
          <<*r<<endl;
   cin.get();
   return 0;
}
```

THRijswijk

find_if

```
#include <iostream>
#include <list>
#include <algorithm>
using namespace std;

class IsPos {
public:
   bool operator()(int i) {
      return i>=0;
   }
};

int main() {
   list<int> l;
   l.push_back(-3);
   l.push_back(-4);
   l.push_back(3);
   l.push_back(4);
   list<int>::iterator r;
   r=find_if(l.begin(), l.end(), IsPos());
   if (r!=l.end())
      cout<<"Het eerste positieve element is:"
          <<*r<<endl;
...
```

THRijswijk

find_if

```
#include <iostream>
#include <list>
#include <algorithm>
#include <functional>
using namespace std;

int main() {
   list<int> l;
   l.push_back(-3);
   l.push_back(-4);
   l.push_back(3);
   l.push_back(4);
   list<int>::iterator r;
   r=find_if(l.begin(), l.end(),
         bind2nd(greater_equal<int>(),0));
   if (r!=l.end())
      cout<<"Het eerste positieve element is:"
          <<*r<<endl;
   cin.get();
   return 0;
}
```

THRijswijk

Functie-objecten

Een functie-object is een object met een operator().

- **Predicate** functie-object met een argument van type T dat true of false teruggeeft.
- **Comparisor** functie-object met twee argumenten van type T dat true of false teruggeeft.
  - equal_to<T> not_equal_to<T>
  - greater<T> greater_equal<T>
  - less<T> less_equal<T>
- **Arithmetic and logical**
  - plus<T> return t1+t2
  - ...

Een Comparisor kan worden omgezet in een Predicate met behulp van een **binder**.
bind2nd(greater_equal<int>(), 0)

THRijswijk

for_each

```
#include <vector>
#include <iostream>
#include <iterator>
#include <algorithm>

using namespace std;

void printDubbel(int i) {
   cout<<i<<" "<<i<<" ";
}

int main() {
   vector<int> v;
   v.push_back(-3);
   v.push_back(-4);
   v.push_back(3);
   v.push_back(4);
   ostream_iterator<int> iout(cout, " ");
   copy(v.begin(), v.end(), iout);
   cout<<endl;
   for_each(v.begin(), v.end(), printDubbel);
   cout<<endl;
   cin.get();
   return 0;
}
```

THRijswijk

Algoritmen

mutating

- **Kopiëren**
  - copy copy_if
- **Bewerken**
  - transform
  - replace replace_if
  - replace_copy replace_copy_if
  - rotate rotate_copy
  - random_shuffle
- **Verwisselen**
  - swap_range
  - reverse reverse_copy
- **Verwijderen**
  - remove remove_if
  - remove_copy remove_copy_if
  - unique unique_copy
- **Diversen**
  - fill fill_n
  - generate generate_n

THRijswijk

transform

```
#include <iostream>
#include <vector>
#include <iterator>
#include <functional>
#include <algorithm>
using namespace std;

int telop(int i, int j) { return i+j; }

int main() {
   vector<int> v, w;
   v.push_back(-3); v.push_back(-4);
   v.push_back(3); v.push_back(4);
   w.push_back(1); w.push_back(2);
   w.push_back(3); w.push_back(4);
   ostream_iterator<int> iout(cout, " ");
   copy(v.begin(), v.end(), iout); cout<<endl;
// Bewerking opgeven met een functie.
   transform(v.begin(), v.end(), w.begin(),
                                 v.begin(), telop);
   copy(v.begin(), v.end(), iout); cout<<endl;
// Bewerking opgeven met std functie-objecten.
   transform(v.begin(), v.end(), w.begin(),
                               v.begin(), plus<int>());
   copy(v.begin(), v.end(), iout); cout<<endl;
   cin.get(); return 0;
}
```

THRijswijk

remove

```
#include <iostream>
#include <vector>
#include <iterator>
#include <algorithm>
#include <functional>
using namespace std;

int main() {
   vector<int> v;
   for (int i(0); i<10; ++i) {
      v.push_back(i*i);
   }
   ostream_iterator<int> out(cout, " ");
   copy(v.begin(), v.end(), out); cout<<endl;
   vector<int>::iterator end(
      remove_if(v.begin(), v.end(),
            not1(bind2nd(modulus<int>(), 2)))
   );
   copy(v.begin(), end, out); cout<<endl;
   copy(v.begin(), v.end(), out); cout<<endl;
   v.erase(end, v.end());
   copy(v.begin(), v.end(), out); cout<<endl;
// ...
```

THRijswijk

Uitvoer

```
0 1 4 9 16 25 36 49 64 81
1 9 25 49 81
1 9 25 49 81 25 36 49 64 81
1 9 25 49 81
```

THRijswijk

Algoritmen

Sorteren e.d.

- **Sorteren**
  - sort
  - stable_sort
  - binary_search
- **Set operations**
  - includes
  - set_union
  - set_intersection
  - set_difference
  - set_symmetric_difference
- **Diversen**
  - ...

THRijswijk

mem_fun

```
#include <iostream>
#include <list>
#include <algorithm>
#include <functional>
using namespace std;

class Hond {
public:
   virtual void blaf() const =0;
...
};

class Tekkel: public Hond { ... };

class StBernard: public Hond { ... };

int main() {
   list<Hond*> k;
   k.push_back(new Tekkel);
   k.push_back(new StBernard);
   k.push_back(new Tekkel);
   for_each(k.begin(), k.end(),
      mem_fun(&Hond::blaf));
...
```

THRijswijk

Galgje

Met gebruik van sets

```
#include <iostream>
#include <string>
#include <set>
#include <algorithm>
using namespace std;

int main() {
   string w("galgje");
   set<char> geraden, letters;
   copy(w.begin(), w.end(),
      inserter(letters, letters.begin()));
   do {
      for (string::const_iterator i(w.begin());
            i!=w.end(); ++i)
         if (geraden.count(*i)) cout<<*i;
         else cout<<'.';
      cout<<endl<<"Raad een letter: ";
      char c;
      cin>>c;
      geraden.insert(c);
   }
   while (!includes(
      geraden.begin(), geraden.end(),
      letters.begin(), letters.end()));
   cout<<w<<" is geraden!"<<endl;
```

THRijswijk

Galgje

Met gebruik van sets en for_each

```
// includes idem

set<char> geraden;

void printLetter(char c) {
   if (geraden.count(c)) cout<<c;
   else cout<<'.';
}

int main() {
   string w("galgje");
   set<char> letters;
   copy(w.begin(), w.end(),
      inserter(letters, letters.begin()));
   do {
      for_each(w.begin(), w.end(), printLetter);
      cout<<endl<<"Raad een letter: ";
      char c;
      cin>>c;
      geraden.insert(c);
   }
   while (!includes(
      geraden.begin(), geraden.end(),
      letters.begin(), letters.end()));
   cout<<w<<" is geraden!"<<endl;
```

THRijswijk

Galgje

Met gebruik van strings

```
#include <iostream>
#include <string>

using namespace std;

int main() {
   string w("galgje");
   string geraden(w.length(), '.');
   do {
      cout<<geraden<<endl
         <<"Raad een letter: ";
      char c;
      cin>>c;
      for (string::size_type i(0);
            i<w.length(); ++i)
         if (w[i]==c)
            geraden[i]=c;
   }
   while (geraden!=w);
   cout<<w<<" is geraden!"<<endl;
   cin.get(); cin.get();
   return 0;
}
```

THRijswijk

Galgje

Met gebruik van strings en transform

```
// includes idem

class Vulin {
public:
   Vulin(char c): c(c) { }
   char operator()(char c1, char c2) {
      return c1==c ? c1 : c2;
   }
private:
   char c;
};

int main() {
   string w("galgje");
   string geraden(w.length(), '.');
   do {
      cout<<geraden<<endl
         <<"Raad een letter: ";
      char c; cin>>c;
      transform(w.begin(), w.end(),
         geraden.begin(), geraden.begin(),
         Vulin(c));
   }
   while (geraden!=w);
   cout<<w<<" is geraden!"<<endl;
```

THRijswijk

Galgje

```
// includes idem
class Raad {
public:
   Raad() {
      cin>>c;
   }
   char operator()(char c1, char c2) {
      return c1==c ? c1 : c2;
   }
private:
   char c;
};

int main() {
   string w("galgje");
   string geraden(w.length(), '.');
   do {
      cout<<geraden<<endl
         <<"Raad een letter: ";
      transform(w.begin(), w.end(),
         geraden.begin(), geraden.begin(),
         Raad());
   }
   while (geraden!=w);
   cout<<w<<" is geraden!"<<endl;
```

Met gebruik van strings en transform

THRijswijk