TDP004 Objektorienterad Programmering
Fö 3 Standardbiblioteket

Standardbiblioteket (STL)

- Standard template library (STL).
- Objektorienterad samling av standardiserade containrar, iteratorer, funktioner, strömfunktioner mm. Platformsoberoende.
- Pga standardisering ger det bra hävstång att lära sig några containers, algoritmer etc.
- Laboration på STL.

Namespaces

Vi kan skriva:

```
std::cout <<variable <<std::endl;
```

eller

```
using namespace std;
cout << variable << endl;
```

- För att markera att en del av de funktioner som vi använder tillhör namespace std. Vi får även fördelen att vi inte behövder skriva std:: före cin/cout.
- Namespaces används för att separera delar av program, och undvika namnkollisioner. Alla containrar heter egentligen std::<container>, men vi kan skriva <container> om vi har specificerat namespace std enligt ovan.
- Använd enbart using namespace i .cc-filen.

Templates

- Templates (mallar) har en viktig funktion i C++.
- Ex:
  - En list/vector innehåller godtycklig data, vilket typ av data det är specificeras med en template T.
  - std::list<T> myList;
  - std::vector<T> myVector

  Där T är någon typ: int, float, yourClass*…
- Allt man behöver göra för att använda en STL-container är att specificera vilken typ den ska innehålla.

Containrar

- ”A container manages a collection of elements”
- #include<containernamn>
- Olika sorter: sekventiella och associativa.
  - Sekventiella: list och vector, deque.
  - Associativa: set/multiset, map/multimap.
- Fokus på de sekventiella: list, vector (och string).
  - Mha dessa implementeras en del andra containrar: stack, kö, prioritetskö.
  - Dequeue väldigt lik vector.
- Interfacen är till stor del standardiserade.

Iteratorer

- En iterator är ett objekt som möjlighet att iterera över/få tillgång till/ta bort/jämföra elementen i en container.
- En iterator representerar en position i en container.
- Inkluderas då man inkluderar containern.
- Funktioner liknande en pekare: *, ++, !=, ==, =, et.c.
- Alla vanliga containerklasser är kompatibla med iteratorer.
- Iteratorer deklareras för en typ av container och en typ av element.

Forward och reverse iterator

- Syntax:
  container<typ>::iterator variabelnamn;
- En forward iterator går från första elementet och ”framåt” mot det sista elementet.
- En reverse iterator går från sista elementet och ”bakåt” mot det första elementet.
- Ex:

```
vector<int> v;
vector<int>::iterator i;
for(i = v.begin(); i != v.end(); i++)        // Från första till sista.
vector<int>::reverse_iterator k;
for(k = v.rbegin(); k != v.rend(); k++)   // Från sista till första.
```

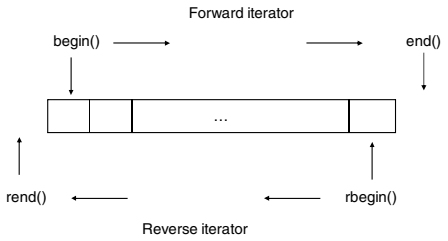
Båda stegas framåt med ++

const interator

- Även iterator kan vara const.
- En const iterator kan inte ändra elementen i den container den används på.
- Ofta användbart, tex då man ska iterera igenom elementen i en container för att skriva ut dem.

  Ex: vector<int>::const_iterator i;
  vector<int>::const_reverse_iterator k;

Översiktlig bild över iteratorer



vector

- #include<vector>
- Används istället för C-array [].
- Likheter med array:
  - Alla element i en följd i minnet.
  - Snabbt tillgång till elementen.
  - Subscript-operator [] kan användas för omedelbar access.
- Skillnad:
  - Vector har dynamisk minnesallokering. Dvs om utrymmet tar slut allokeras automatiskt mer! Minnesutrymmet fördubblas vid varje ny allokering.
- Snabbast att lägga till element sist.
- Långsamt att ta bort element på alla platser utom sista pga kopiering av övriga element.

Exempel med vector och iterator

```
std::vector<int> v;                 //Skapa tom vector
std::vector<int>::iterator i;       //Deklarera iterator.
v.push_back(4);                     //Lägg till element sist.
v.push_back(1);

//Iterera över elementen mha iterator.
for(i = v.begin(); i != v.end(); i++)
{
   std::cout<<(*i)<<" ";            //Skriv ut elementet.
}
->
4 1
```

Exempel med vector och iterator, forts.

- Mha en iterator och for/while-loop kan vi se till att vi inte läser utanför vectorn.
  Detta är skäl nog att gå över från array till vector.

```
for(std::vector<int>::iterator i = v.begin(); i != v.end(); i++)
   {
     cout << (*i) << " ";
   }
```

- Vi kan använda en vanlig for-loop och en (unsigned) int för att iterera, men då tappar vi fördelar, ingen felkontroll i detta fall.

```
for(unsigned int j = 0; j < v.size(); j++)
   {
     cout << v[j] << " ";
   }
```

list

- #include<list>
- Dubbellänkad lista.
- Elementen potentiellt utspridda i minnet.
- Ingen omedelbar access.
- Samma tid för att lägga till/ta bort element i början och slutet på listan.

```
list<int> l;
l.push_back(1);
l.push_front(2);
->2 1
```

Exempel med list

- Ta bort alla element med value:

```
list<int> l1;
…                    //Lägger till element.
l1.remove(value);
```

- Flytta alla element från l2 (gör l2 tom efteråt) och placera dem först i l1:

```
list<int> l2;
l1.splice(l1.begin(), l2,
          l2.begin(), l2.end());
```

string

- string sparar bokstäver och andra tecken.
- STL string, inte samma som C-string (char*/ const char*), men omvandlingsfunktioner finns.
- STL string är att föredra.
- En del funktionalitet liknande C-string.
- Funktioner för att lägga till tecken, jämföra, access, ta bort mm finns.

string, forts.

- string s1(”Add more”);
- string s2 = ”characters”;
- string s3 = s1 + s2; // ”Sätta ihop” string
- string s4 = s1.find(m) // s4 = ”more”
- s1[1] = ’n’; // s1 = ”And more”

Map/multimap

map<key, value> mapExample;

- I multimap kan det finnas kopior av nyckeln.
- En associativ container sparar element i nyckelordning.
- Implementeras normalt med ett träd.

map, pair

- Associativ container
  - Ett värde associeras med en nyckel. Dessa bildar *pair*, vilket är det grundläggande elementet.
- Värdet får ändras när paret existerar.
- Nyckeln får inte ändras.
  - Om man vill ändra nyckeln får man ta bort det gamla paret och lägga till ett nytt par.

map, insert

- Lägg till element mha insert.
- Ex:

```
string cplusplus = ”C++”;
string java = ”Java”;
map< string, int> pCount;
pCount.insert( make_pair(cplusplus, 2) );
pCount.insert( make_pair(java, 8) );
```

Inget händer vid:

```
pCount.insert( make_pair(cplusplus , 3));
```

eftersom nyckeln C++ redan finns.

map, [] och find

- [] finns normalt inte till associativa containrar, map är undantag.
- pCount[”C++”] = 4;
  Sätter värdet associerat med nyckeln C++ till 4. Om inte nyckeln C++ hade funnits hade ett pair skapats och värdet satts.
- find returnerar en iterator till elementet som har den nyckel som man söker efter. Om elementen inte finns returneras en iterator till end().
  Ex:

```
map< string, int>::iterator i = pCount.find(”java”);
if(i != pCount.end())
{
      cout << i->second << endl;
}
```

-> 8

- Snabbt att använda find pga intern trädstruktur.

map, erase och iteration

- pCount.erase(”Java”);
  - Tar bort alla element med nyckeln Java.
  - Returnerar antalet element som togs bort (0,1 för map)
- Iteration

```
map< string, int>::iterator i = pCount.begin();
while( i != pCount.end() )
{
   cout << (*i)->second << endl;
   i++;
}
```

Funktioner

- #include<algorithm>
- STL tillhandahåller funktioner för sortering, sökning efter det minsta/största elementet, räkna antalet element som uppfyller ett visst krav...
- Uppdelade i icke-modifierande, modifierande, ”removing”, muterande, sorterande.
- Många funktioner använder och returnerar iteratorer.
- Byggstenar till egna funktioner.

Funktioner

- Dels finns funktioner som är oberoende av containrar, samt containrarnas egna funktioner som enbart fungerar på en viss container.
- Orsaken är effektivare/kraftigt skild implementation mellan olika containrar.

Sortering

- STL har inbyggda funktioner för sortering.
- Exempel sortering av vector:
  sort(startiterator, slutiterator, binärpredikat);
  - Sorterar alla element inom intervallet [startiterator, slutiterator).
  - binärpredikat är en funktion som anger kriterier för vad elementen ska sorteras på. Måste generellt tillhandahållas av programmeraren.
- Motsvarande finns för list.

Sökning

- Även inbyggda funktioner för sökning finns:
- iterator find_if(startiterator, slutiterator, unärpredikat)
  - Söker alla element inom [startiterator, slutiterator), och kontrollerar mha unärpredikat.
- iterator find(startiterator, slutiterator, const T& värde);
  - För att söka efter ett element som har ett visst värde.
- Returvärdet är en iterator till det första element som uppfyller kravet/ unärpredikatet är uppfyllt.

Sökning, forts.

- Ex:
  - Sök efter det första element som har värdet 4:

```
list<int> l1;
l1.push_back…
list<int>::iterator element;
element = l1.find(l1.begin(), l1.end(), 4);
if(element != l1.end())
{
   cout <<"Element existed."<<endl;
}
```

Ex: Ta bort kopior från vector

- vector<int> v;
- v.push_back(1)
- v.push_back(3)
- v.push_back(1)
- vector<int>::iterator newEnd;
- newEnd = unique(v.begin(), v.end());
- unique flyttar på direkt efterföljande kopior och returnerar en iterator till det nya logiska slutet på containern. Inga efterföljande kopior i det här fallet -> sortera först.
- sort(v.begin(), v.end());
- newEnd = unique(v.begin(), v.end()); //newEnd pekar kanske på 3.
- v.erase(newEnd, v.end()); //tar bort all element inom[newEnd, v.end()) , dvs de flyttade elementen.

Varför iteratorer?

- En av poängerna med STL är att enkelt kunna byta container. Kom ihåg att interfacen till stor del är standardiserade.
- Många av funktionerna kräver iteratorer, t.ex. find, sort, insert, erase mfl.
- Exempel

```
vector<int> v;
for(int i = 0; i < v.size(); ++i)        /Fungerar.
{
   …
}

list<int> v;
for(int i = 0; i < v.size(); ++i)        //Fungerar ej. < finns ej för list.
{
   …
}
```

Varför iteratorer? (forts.)

- Använd iteratorer istället.

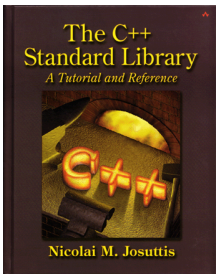
```
vector<int> v;
vector<int>::iterator i;
for(i = v.begin(); i != v.end(); ++i)
{
   …
}
```

Byt vector mot list. for-loopen fungerar nu utan problem.

```
list<int> v;
list<int>::iterator i;
for(i = v.begin(); i != v.end(); ++i)
{
   …
}
```

Mera information om STL

- Nicholai M. Josuttis
  The C++ Standard Library:
  A Tutorial and Reference.
  Addison-Wesley 1999.
- Rekommenderas!



Sammanfattning

- Introducerat standardbiblioteket (STL).
- Sekventiella containrar som tex vector, list och string.
- Iteratorer används för iteration, access etc.
- Inbyggda funktioner för sökning, sortering och mycket mer.
- **STL är stort och kraftfullt, lär er och använd det!**