EDAF30 – Programmering i C++

*2. Tecken och texter*

Sven Gestegård Robertz
*Datavetenskap, LTH*

2015

## Tecken

### char

```
char ch = 'A';
```

### sizeof(char) = 1 byte (ej 16-bitars tecken som i Java)

```
char c1, c2;
c1 = 'A';
c2 = 106; // Samma som 'j'
cout << c1 << c2 << '!' << endl;
```

1 byte ej nödvändigtvis en *oktett* (8 bitar).



## Strängar

Ordet "sträng" (*eng:string*) kan betyda två saker i C++.

### standardklass i C++: std::string

```
string s = "Hej";
```

### C-strängar: null-terminerad char[]

```
char str[] = "Hej";
```



## C-strängar ("Teckenfält")

### Lagring i minnet

- *null-terminerad*: tecknet \0 markerar slutet på strängen
- `char namn[]= "Nils";  // längd = 5 bytes`

  är ekvivalent med

  ```
  char namn[] = {'N', 'i', 'l', 's', '\0'};
  ```

- | adress | data |
  |---|---|
  | namn | N |
  | namn+1 | i |
  | namn+2 | l |
  | namn+3 | s |
  | namn+4 | \0 |

### Access av element

```
cout << namn[1] << namn[3];
```

is



## C-Strängar

### char[] är inte samma som char *

Istället för deklarationen

```
char str[] = "12345";
```

ser man ofta

```
const char *str2 = "12345";
```

men

```
sizeof(str) == 6  // 6 element      sizeof(str2) == 8  //!!!
for(char c: str)  // OK!            for(char c: str2)  // Error!
   ...                                ...
```

NB! Generellt: antal element i en array av T:

```
T array[] {...};
...
size_t n = sizeof(array)/sizeof(T);
```



## Teckenkodning

- 7-bitars ASCII: fungerar i alla (icke-antika) system
- många inkompatibla 8-bitars-kodningar
- Unicode-standarden
  - UTF-8 - den vanligaste teckenkodningen
  - UTF-16 (char i Java)
  - UTF-32 - alla tecken är lika stora

### UTF8: ett tecken representeras som en eller flera byte

- En UTF-8-kodad sträng kan lagras i en char[].
- 7-bitars ASCII-tecken lagras i en byte

### wchar_t: C++ "multi-byte characters"

- C++-standarden specificerar inte storleken (jfr char i Java),
  - oftast 32 bitar (UTF-32 fungerar)
  - Microsoft C++: 16 bitar

## Teckenkodning
### Åttabitars tecken i Windows

**Två olika 8-bitars-tabeller**

- ISO-8859-1 (Latin-1) (Se Appendix B)
  - Windows code page 28591 (ISO-8859-1) (Latin-1)
  - Windows code page 1252 (Windows Latin-1)
- 850 (IBM-ASCII): Används bara i DOS



## Svenska tecken under Windows

**Ändra kodning i konsollfönstret**

```
#include <iostream>
#include <cstdlib>

using namespace std;

int main()
{
   system("chcp 1252 >nul 2>nul");

   char ch = '7';
   int code, val;
   code = ch;
   val = ch - '0';
   cout << "Siffran " << ch << " har koden " << code
      << " och värdet " << val << endl;
}
```



## Svenska tecken under Windows

**med `iodos.h`, se förra årets kurshemsida/extramaterial**

```
#include <iostream>
#include "iodos.h"

using namespace std;

int main()
{
   dos_console();

   char ch = '7';
   int code, val;
   code = ch;
   val = ch - '0';
   cout << "Siffran " << ch << " har koden " << code
      << " och värdet " << val << endl;
}
```



## Specialtecken
### *control characters, non-printable characters*

**Escape-sekvenser (s. 71 i boken)**

```
\n   (ASCIIkod 10 = newline, line feed, LF)
\a   (ASCIIkod  7 = bell)
\t   (ASCIIkod  9 = tab)
\r   (ASCIIkod 13 = carriage return, "vagnretur", CR)
\b   (ASCIIkod  8 = backspace)
\nnn (tecken med ASCIIkod nnn (oktalt))
\xnn (tecken med ASCIIkod nn (hexadecimalt)
\\   (backslash)
\"   (citationstecken)
\'   (apostrof)
```

Radslut inkompatibelt mellan olika system

Unix : LF

Windows : CR LF

Äldre mac : CR



## Teckenhantering

**In- och utmatning av tecken**

```
char c;

cout << c        // Skriver ut tecknet c
cout.put(c)      // Skriver ut tecknet c

cin >> c         // Läser in nästa icke-vita tecken till c
cin.get(c)       // Läser in nästa tecken till c

cin.peek()       // Returnerar nästa tecken som förblir oläst
cin.ignore()     // Läser nästa tecken men slänger det
```



## Inmatning

**Exempel 1**

```
#include <iostream>
using namespace std;
int main(){
   char namn[30];
   cout << "Vad heter du? ";
   cin >> namn;
   cout << "Hej, " << namn << "!" << endl;
}
```

Vid körning:

```
Vad Heter du? Lars Olof Persson
Hej, Lars!
```

Byt ut `cin >> namn` mot `cin.getline(namn,30)`.

`cin.getline(var,antal)` läser $\leq$ *antal* tecken, *inkl. null*

## Inmatning

### Exempel 2

```
#include <iostream>
using namespace std;
// Kryptiskt program
char kod[33]="qwertyuiopasdfghjklzxcvbnm012345";
int main()
{
   char c;
   cout << "Avslutning sker med Ctrl-z" << endl;
   while (cin.get(c)) {
      cout << kod[c%32];
   }
   cout << endl;
}
```

Använder

```
istream& istream::get(char& c);}
```



## Inmatning
### Varnande exempel

### Den lästa strängen får inte plats i `x`

Satserna

```
char z[] {"zzzz"};
char y[] {"yyyy"};
char x[5];

stringstream sin{"aaaaaaaaaaaaaaaaa bbbbbb"};
sin >> x;

cout << x << " : " << y << " : " << z << endl;
```

Ger vid körning (på min dator):

```
aaaaaaaaaaaaaaaaaa : aaa : zzzz
```

- C-strängar ger inget skyddsnät
- läsningen till `x` har skrivit över (en del av) `y`
- `getline()` är säkrare



## Inmatning

### Exempel 3

```
#include <iostream>

using namespace std;

int main()
{
   string s;
   int tal = 0;
   cout << "Ange ett tal: ";
   cin >> s;
   for (char ch : s) {  // C++11
      tal = 10 * tal + ch - '0';
   }
   cout << "Talets värde: " << tal << endl;
}
```



## C-strängar – Funktioner

### In- och utmatning av tecken

```
#include <cstring>  // Bibliotek att inkludera

strcpy(s,t)     // Kopierar t till s
strncpy(s,t,n)  // Variant med max n tecken

strcat(s,t)     // Lägger till t till slutet av s
strncat(s,t,n)  // Variant med max n tecken

strlen(s)       // Returnerar längden av s

strcmp(s,t)     // Jämför s och t
strncmp(s,t,n)  // Variant med max n tecken
// s<t, s==t, s>t ger resp. <0, =0, >0
```

Osäkra, undvik!



## Initiering och tilldelning

### Initiering

```
char s[4]= "abc"; // OK
char s[] = "abc"; // OK
char s[3]= "abc"; // Inte OK (pga \0)
```

### Tilldelning

```
s = "def";       //Felaktigt!
s[0]='d'; s[1]='e'; s[2]='f'; // OK

// Bättre variant:
strncpy(s, "def", 4);  // kräver #include <cstring>
```



## Strängkopiering
### Varnande exempel

Satserna

```
char s[20];

strncpy(s, "abc", 4);
cout << s << endl;

strncpy(s, "kalle anka", 20);
cout << s << endl;

strncpy(s, "def", 3);
cout << s << endl;
```

ger utskriften

```
abc
kalle anka
defle anka
```

Satserna

```
int data[] {558646598, 65, 66};
char x[16];
char t[30] {"test"};

strncpy(x,"abcdefghijklmnop",16);
strcpy(t,x);
cout << t << endl;
```

ger utskriften

```
abcdefghijklmnopFEL!A
```

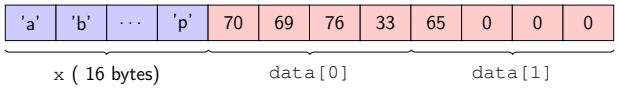
**Tänk på att**

- `strncpy` måste ha plats till avslutande `\0`.
- `strcpy` kopierar tills den hittar ett `\0` i `src`.

## Strängkopiering
### Varnande exempel: förklaring

```
int data[]{558646598, 65, 66};
char x[16];
```

- `int data[]` tolkas som `char[]`.
- representation i minnet




x ( 16 bytes) data[0] data[1]

Hexadecimal representation:

$$558646598_{10} = 214c4546_{16}$$
$$65_{10} = 41_{16}$$

Byte-ordning: *little-endian*

| hex | ASCII) | dec |
|---|---|---|
| 46 | F | 70 |
| 45 | E | 69 |
| 4c | L | 76 |
| 21 | ! | 33 |
| 41 | A | 65 |
| 0 | \0 | 0 |
| 0 | \0 | 0 |
| 0 | \0 | 0 |
| ... | ... | ... |



## Standardklassen `std::string`

Enklare hantering än med `char[]`:

- liknar `java.lang.String`
- har skyddsnät

### Exempel

```
string s1("abc");
string s2="def";
string s3;

cout << s3.size() << endl;

s3 = "ghij";
cout << s3.size() << endl;

s3 = s1 + s2 + s3 + "klmnop";
cout << s3.size() << endl;

s3 += "qrstuvwxyz";;
cout << s3.size() << endl << s3 << endl;
```

```
0
4
16
26
abcdefghijklmnopqrstuvwxyz
```



## Standardklassen `std::string`

### Operatorer på `string`

```
#include <string>

s = t;              // Tilldelning
s.assign(t);        // Tilldelning
s = s + " " + t;    // Konkatenering

s<t, s==t           // Jämförelse
s[k]                // Indexering, hämta eller sätta värdet
s.at(k)             // Indexering med indexkontroll

cout << s << t;     // Utskrift
cin >> s;           // Inmatning av ord
getline(cin, s);    // Inmatning av hel rad
```



## Standardklassen `std::string`

### Funktioner (metoder) i `string`

```
s.substr(k,n)
s.erase(k,n)
s.clear()
s.size()
s.append(t)
s.insert(k,t)
s.replace(k,m,t)
s.find(t)
```



## Sekvenser i C++

- En *sekvens* är en följd av element
- `std::vector`: variabel storlek, $O(1)$ indexering
- `std::array`: fast storlek, $O(1)$ indexering, lika effektiv som C-array, men säkrare
- `std::list`: variabel storlek, $O(1)$ insättning och borttagning, långsam ($O(n)$) indexering
- `std::deque`: variabel storlek, (effektivare än vector), $O(1)$ insättning och borttagning i början och slutet, $O(1)$ indexering.

Med *indexering* avses *random access*



## Sekvenser: exempel
`std::vector`

### Initiering

```
vector<int> a{1,2,3,4,5}; // som förväntat
vector<int> b{5};         // {5}          ett element, 5
vector<int> c(5);         // {0,0,0,0,0}  storlek=5
```

### Initiering från C-array

```
int fib[]  {1,1,2,3,5,8,13,21};
vector<int> vfib(begin(fib), end(fib));

for (int i : vfib) {cout << i << " ";}
cout << endl;

char str[] = "Hello World!";  // 13 tecken inklusive \0
vector<char> v2(str, str+13); // "one past the end"
```

- iteratorer: mer om detta senare
- `begin()` och `end()` kan bara användas i det *scope* där `fib` deklarerats

## Sekvenser: exempel
`std::vector`

```
void print_vector(vector<int>& v)
{
   for(auto x : v)
       cout << x << " ";
   cout << endl;
}
```

### vector::push_back

```
vector<int> v; // tom vector

v.push_back(1);
v.push_back(2);
v.push_back(3);
print_vector(v);

v.clear();
v.push_back(5);
print_vector(v);
```

```
  1 2 3
  5
```

### tilldelning

```
vector<int> v{1,3,5,7,9};
vector<int> v2(10);

print_vector(v);
print_vector(v2);

v2.at(4) = 7; // v2[4] (säker)
print_vector(v2);

v2 =  v;
v = {1,2,3};
print_vector(v);
print_vector(v2);
```

```
  1 3 5 7 9
  0 0 0 0 0 0 0 0 0 0
  0 0 0 0 7 0 0 0 0 0
  1 2 3
  1 3 5 7 9
```



## Sekvenser: exempel
`std::vector`

### Skyddsnät: `vector::at()` resp `[]`

```
vector<int> a {0,1,2,3,4,5,6,7,8,9};

cout << "a[10]: " << a[10] << endl;

cout << "a.at(10): " << a.at(10) << endl;
```

```
a[10]: 33
terminate called after throwing an instance of 'std::out_of_range'
what():  vector::_M{range_check: __n (which is 10) >= this->size()
  (which is 10) }
```



## Sekvenser: exempel
`std::array`

### Exempel

```
array<char,12> a1 {"Hello Again!"};
```

```
error: initializer-string for array of chars is too long
```

### Exempel

```
array<char,13> a1 {"Hello Again!"};

for (char ch : a1) {
   cout << "[" << c << "]";
}
cout << endl;
```

```
[H][e][l][l][o][ ][A][g][a][i][n][!][]
```

NB! utan avslutande \0 blir längden 12:

```
std::array<char,12> a2{'H', 'e', 'l', 'l', 'o', ' ',
                       'A', 'g', 'a', 'i', 'n', '!'};
```



Vi har talat om

- Tecken
- C-strängar
- `std::string`
- Några standard-datatyper

## Nästa föreläsning:

Vi kommer att gå igenom

- Funktioner
- Separatkompilering