Stränghantering med C++

Datatypen char

Används för tecken.
Tecken kan vara bokstäver och siffror, även &, ", |, £ är exempel på tecken. Tecken som inte syns (white space) som används för att formatera texten, till exempel tecken för radbrytning, tab. m.fl. använder också datatypen char. Dessa kallas styrtecken.

char kan lagra heltal. Ett bestämt heltal används för varje tecken. Detta är bestämt i ASCII-tabellen.

Datatypen char använder normalt en byte per tecken i datorns minne. En byte är 8 bitar. Endast 7 bitar används i Windows. Åttonde biten är av historiska skäl reserverad för kontroll av byten (kontrollbit). Extended (utvidgad) ASCII använder alla åtta bitarna, då får våra å, ä och ö också plats. Men i vår kurs så är vi begränsade till engelska bokstäver.

EX:

```
char bokstav = 107;
char bokstav2 = 'k';  // OBS! enkelfnuttarna

cout << bokstav << " " << bokstav2 << endl;

/*
Utskriften blir:
k k
*/
```

Enkelfnuttarna ' ' före och efter tecknet talar om att vi vill lagra ett visst tecken i variabel av typen char.

char lagrar heltal, så vad händer om vi skriver:

```
char c = 'k' + 1;

cout << c << endl;

// Utskriften blir l, se ASCII-tabellen
```

Vi kan förstås göra så här för att skriva ut mer än en bokstav:

```
char c1='F', c2='a', c3='r', c4=' ', c5='r', c6='o',c7='r';

cout << c1 << c2 << c3 << c4 << c5 << c6 << c7 << endl;

// Utskrift: Far ror
```

c-strängar (c-string)

Detta blir ganska omständigt. I stället kan man använda c-strängar. Tänk på dessa som en variabel som kan innehålla många olika värden. Ett "fack" för varje värde.

F	**a**	**r**		**r**	**o**	**r**	

När vi vill deklarera en sådan här c-sträng, med plats för åtta tecken skriver vi så här:

```
char str[8];
```

En char sträng kan få sitt värde samtidigt som den deklareras

```
char str2[8] = "Far ror";

cout << str2;

// Skriver ut Far ror

char str3[] = "Far ror";

cout << str3;

// Skriver ut Far ror

/*
åttan kan utelämnas, c++ räknar själv ut hur många "fack" det
ska vara
*/
```

Varför åtta fack?
C++ lägger själv till '\0' (nolltecknet, ASCII-kod 0) sist. Vid utskrift med

```
cout << str3;
```

skrivs alla tecken fram till 0-tecknet ut. (fram till att '\0' tecknet hittas)

Tyvärr kan man inte tilldela en c-strängs variabel nya värden på ett enkelt sätt. Därför har en modernare klass utvecklats. Den kallas c++ - strängar. Biblioteket för C++ - strängar måste inkluderas för att programmet ska fungera.

```
#include <string>  /* I Dev-C++ behövs det inte, den är redan inkluderad */
```

C++ strängar deklareras så här:

```
string s;
string s1 = "Far ror";
```

En C++ - sträng kan ges nya värden under tiden programmet körs. Den anpassar sig automatiskt till rätt längd (rätt antal "fack").

EX:

```
string namn;

namn = "Ingemar";

cout << namn;

namn = "Eva-Stina";

cout << namn;
```

Fördelar och nackdelar med C-strängar och C++ - strängar

C++ strängen är modernare, mer lätthanterlig och ger programmeraren större frihet. C-strängen är mer resurssnål vad gäller minnesåtgång och processorskraft. Vanligtvis saknar det betydelse att den är mer resurssnål.

In och utmatning till/från programmet (I/O)

```
cout << "hejsan" << 4 << variabel << variabel1 << endl;
```

Tänk att cout är skärmen, << (utmatningsoperatorn) matar ut, i tur och ordning, i en lång rad, till skärmen.

```
int i;
cin >> i;
```

Tänk att cin är tangentbordet (standard inmatningsenhet, vald av operativsystemet) dina tangenttryckningar lägger sig efter varandra i en rad. Denna "rad" är en buffert som laddas upp och kan skickas iväg med Retur. Den kallas **tangentbordsbuffert**. Först när du trycker Retur (Enter) åker hela raden iväg, i den ordning du tryckt på tangenterna.

EX 1:

```
char namn2[100];
cout << "Vad heter du?: ";
cin >> namn2;

// När du trycker Retur far dina tecken in i variabeln namn;

cout << namn2;

// Det du skrivit in skrivs ut på skärmen.
```

EX 2:

```
string namn3;
cout << "Vad heter du?: ";
cin >> namn3;

// När du trycker Retur far dina tecken in i variabeln namn;

cout << namn3;

// Det du skrivit in skrivs ut på skärmen.
```

Vad händer om du skriver både för och efternamn? (Egon Kjerrman)

Det som händer är att **cin** läser in tills att första blanka tecken hittas, det vill säga fram tills att mellanslaget mellan för- och efternamn hittas. Efternamnet ligger kvar i **tangentbordsbufferten**. Kör exempel 1 och 2 direkt efter varandra och använd för och efternamn med mellanslag emellan, så får du se.

Konsten att rensa tangentbordsbufferten

Vill du bli av med det som finns i tangentbordsbufferten, kan du ta bort tecken med

```
cin.ignore(100, '\n');
```

Detta tar bort alla tecken, högst 100 stycken och slutar innan 100 stycken tecken om returtecknet hittas. Pröva att placera in koden mellan exempel 1 och 2.

Men om man vill kunna mata in med mellanslag?

Det kan ju vara praktiskt att kunna mata in både för och efternamn på en gång. Du kan göra så här.

Med C-strängar

EX 3

```
char namn4[30];
cin.get(namn4, 30);
/* Högst 29 tecken kan matas in, plats måste finnas för '\0'
på slutet */
```

EX 4

```
char namn5[30];
cin.getline(namn5, 30);
/* Högst 29 tecken kan matas in, plats måste finnas för '\0'
på slutet */
```

Skillnaden mellan **cin.get(namn4, 30);** och **cin.getline(namn5, 30);** är att cin.getline(namn5, 30); plockar bort '\n' (returtecken) från slutet av inmatningsbufferten.

När behöver du använda **cin.ignore(100, '\n');** ?

Med C++ strängar

```
string str2;

getline(cin, str2);
```

Så här kan du hantera strängar med mellanslag, när du använder **string**.

Konsten att komma åt enskilda tecken

Både C-strängar och C++ - strängar kan man komma åt med index. Skriva ut eller ändra enskilda tecken.

EX 5

```
char test[] = "Hejsan";

cout << test << endl; // skriver ut Hejsan

cout << test[0] << endl; // skriver ut H

test[1] = 'a';

cout << test << endl; // skriver ut Hajsan
```

EX 6

```
string s = "Hejsan";

cout << s << endl; // skriver ut Hejsan

cout << s[0] << endl; // skriver ut H

s[1] = 'a';

cout << s << endl; // skriver ut Hajsan
```

EX 7 (Också möjligt!)

```
char c = 'P';
string s = "Hej";
s[0] = c; // s får värdet "Pej"
```

Det rekommenderas att använda C++ - strängar om det inte finns ett bra skäl att använda C-strängar!