Lathund 3.0 för programmering i C++

Datatyper

bool - har antingen värdena **true** eller **false**
int - ett 32-bitars **heltal**. (-2^31) till (2^31 - 1), ca 2*10^9
long long - ett 64-bitars **heltal**. (-2^63) till (2^63 - 1), c.a 9*10^18.
double - ett **decimaltal** (med begränsad nogrannhet)
string - en **textsträng**, t.ex. “herp derp” (notera dubbla citattecken)
char - en **bokstav**, t.ex. ‘a’, ‘b’, ‘0’, ‘!’ (notera enkla citattecken)

Syntax

Deklarera en variabel:

```
datatyp variabelnamn;
int x;
vector<bool> y;
double z;
```

Tilldela värden:

```
variabelnamn = värde;
x = 10;
y[5] = 10; //För vectors
double z = 5+6; //Kombinera deklaration och tilldelning
```

Operationer:

```
6%5
y = (z * 3);
y += 1; //ökar med 1
y++; //ökar också med 1
```

Main-funktionen:

```
int main(){
      //...kod här...
}
```

Includes:

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

Vanliga filer: vector, set, map, string, stack, queue, algorithm, cmath, iostream

If-satser

```
if(<booleskt uttryck>){
	//...kod om uttrycket är sant...
} else if(<annat booleskt uttryck>){
	//...kod om detta uttryck är sant men inte det första...
} else {
	//...kod om inget av de tidigare uttrycken var sanna...
}

if(day == “wednesday”){
	goToPaul();
}  else if(day == “tuesday”) {
	bakeCookies();
} else {
	cry();
}
```

For-loopar:

```
for(<initalisering>; <villkor>; <uppdatering>){
	//..kod om uttrycket är sant...
}
for(int i = 0; i < 10; i++){
	//..kod som kommer köras 10 gånger..
}
```

While-loopar:

```
while(<villkor>){
	//..kod som körs så länge uttrycket är sant...
}

while(hour != 7){
	hour = (hour + 1);
	hour = hour%24;
}
```

Deklarera funktioner:

```
returtyp namn (argumentlista){
	//...kod här...
	return <returnvärde>;
}

void derp(int x, int y){
	if(x == 5){
		return;
	}
	herp();
}
int getRandomNumber(){
	return 4; //Determined by fair die roll
}

void plusett(int& x){
	x += 1; // ändrar x, eftersom x var en referens
}
```

Glöm inte: funktioner kan bara användas efter deras deklaration.

Anropa funktioner:

```
aVector.push_back(52);
doSomethingEeeeevil();
```

Glöm inte: när du anropar en funktion kommer en kopia av argumentet skickas om du inte använder en **referens**: `void herp(int &parameter)`

STL

vector

En vector är en array som kan växa och krympa.
`vector<int> namn;` - skapar en tom vector som håller intar
`vector<double> namn(4);` - skapar en vector fylld med 4 objekt och innehållet [0.0, 0.0, 0.0, 0.0]
`vec.push_back(värde);` - lägger till ett element i slutet av vectorn
`vec.size();` - returnerar antalet element i vectorn
`vec[x];` - returnerar elementet på plats x i vectorn
`vec[y] = värde;` - tilldelar värde till en plats i vectorn

```
vector<int> hej(4);          //[0, 0, 0, 0]
hej[0]++;                    //[1, 0, 0, 0]
hej.push_back(hej.size());   //[1, 0, 0, 4]
cout << hej[4] << endl;      //skriver ut 4
```

Glöm inte: vektorer är **NOLL-INDEXERADE**.
Om en vektor har storleken 4 kan du läsa värdena 0, 1, 2, 3!!!!

set

Ett set är en sorterad mängd med unika värden.
`set<int> namn;` - skapar ett tomt set
`set.insert(värde);` - lägger in ett element i set
`set.erase(värde);` - tar bort ett värde ur ett set
`set.find(värde);` - om värdet finns returneras iterator till värdet. Annars set.end(). Dvs ett värde existerar om och endast om set.find(värde) != set.end()
`set.size()` - returnerar antalet element i settet

```
set<int> hej2;
hej2.insert(-1);

if(hej2.find(-1) != hej2.end()){
	cout << “-1 finns i set” << endl;
}
if(hej2.find(2) != hej2.end()){
	cout << “2 finns i set” << endl;
}
```

kommer bara skriva ut “-1 finns i set”

```
hej2.erase(-1); // nu är hej2 tom
```

map
En map är en “ordboksstruktur” som mappar en mängd till en annan. Som en vector men man får använda vilka index som helst.
`map<string, string> m;` - skapar en ny map
`m[x];` - hämtar “definitionen” av “ordet” x i vår map
`m[x] = y;` - ändrar “definitionen” av “ordet” x till y

```
map<string, int> s;
s[“A”] = 1;
cout << s[“A”] << endl; // skriver ut 1
```

queue
En kö av saker.
`queue<double> q;` - skapar en ny kö
`q.front();` - returnerar det **första** värdet i kön
`q.pop();` - tar bort det **första** värdet i kön
`q.push(värde);` - lägger till ett värde **sist** i kön
`q.size();` - returnerar antalet element i kön

stack
En kö av saker där man istället plockar bort det senast insatta elementet
`stack<string> s;` - skapar en ny stack
`s.top();` - returnerar det **översta** värdet i stacken
`s.pop();` - tar bort det **översta** värdet i stacken
`s.push(värde);` - lägger till ett värde **överst** i stacken
`s.size();` - returnerar antalet element i stacken

string
En sträng är en samling av tecken -- fungerar som en vector<char>
string namn; - skapar en tom sträng
`string namn = “derp”;` - skapar strängen med innehållet “derp”
`namn[2];` - returnerar det tredje tecknet (glöm inte att det är nollindexerat!) i strängen
`namn[2] = ‘z’;` - ändrar det tredje tecknet i strängen till ‘z’.
`namn.push_back(‘a’);` - lägger till teckent ‘a’ på slutet av strängen
`namn.size();` - returnerar längden på strängen

pair
Ett par håller två olika saker i ett enda.
`pair<int, string> par(512, “the game”);` - skapar ett nytt par med värdena 512 och “the game”
`par.first` - returnerar det första värdet i paret
`par.second` - returnerar det andra värdet i paret

cin och cout
cin och cout används för att läsa in och skriva ut data.
`cin >> var1 >> var2;` - läser in värden i var1 och var2
`cout << var1 << var2;` - skriver ut värdena var1 och var2 **utan mellanslag**
`cout << var1 << " " << var2;` - skriver ut värdena var1 och var2 **med mellanslag**
Glöm inte: för att skriva ut en ny rad ska du skriva ut **endl**

sort
`sort(namn.begin(), namn.end());` - sorterar en **vector**

max, min, abs
`max(x, y)` och `min(x, y)` räknar ut max och min av två värden
`abs(x)` räknar ut absolutbeloppet av ett tal

<cmath>

`round``(tal);` - avrundar ett decimaltal till närmsta heltal
`ceil``(tal);` - avrundar ett decimaltal till närmsta minst lika stora heltal
`floor``(tal);` - avrundar ett decimaltal till närmsta heltal som är högst lika stort
`sqrt``(tal);` - beräknar kvadratroten av ett tal
`sin``(x);` - räknar ut sinus av **x** angivet i **radianer**
`cos``(x);` - samma som ovan, fast cosinus
`exp``(x);` - beräknar e^x
`log``(x);` - beräknar ln(x)
`M_PI` - en konstant som är definerad till pi

Lista över operationer

Addera: **+**
Subtrahera: **-**
Multiplicera: *****
Dividera: **/**
Ta fram rest: **% (modulo)**
Logiskt "och": **&&**
Logiskt "eller": **||**
Logiskt "inte": **!**
Mindre än: **<**
Större än: **>**
Likhet: **==**
Inte likhet: **!=**
Mindre lika med: **<=**
Större lika med: **>=**

Bitoperationer

Heltal representeras internt som tal i bas 2 (alla siffror är 1 eller 0). Om en bit (“siffra”) är 1 säger vi att biten är *satt*.

`x & y` - gör ett binary and (båda bitar måste vara satta)
`x | y` - gör ett binary or (någon av bitarna måste vara satta)
`x ^ y` - gör ett binary xor (exakt en bit måste vara satt
`~x` - gör en binary negation (inverterar alla bitar)
`x << y` - flyttar alla satta bitar i talet x med y positioner till vänster
`x >> y` - flytter alla satta bitar i talet x med y positioner till höger

För att kolla om biten X är satt: `if(tal & (1<<X))` ...
För att sätta biten X: `tal |= (1 << X);`
För att ta bort biten X: `tal &= ~(1 << X);`
För att “toggla biten” (sätta biten till dens invers): `tal ^= (1 << X);`
För att generera alla delmängder av de satta bitarna:

```
for(int x = tal; x != 0; x = (x-1)&tal)
```

Denna metod missar dock talet 0, som du måste behandla separat.