C++ en överblick

C++ EN ÖVERBLICK

INTRODUKTION TILL C++

Historia och bakgrund

Standarder

C++ i relation till C och ANSI-C

Reserverade ord

Operatorer i C++

Kommentarer i C++
- Exempel

Deklarationer i C++
- Exempel
- In- och utmatning i C++ (streams)
- cin, cout och cerr
- Exempel med cout
- Exempel med cin
- Exempel med cerr

Dynamisk minneshantering i C++
- Felhantering
- Newhandler
- delete
- Referensvariabler
- Exempel
- Referens som returvärde
- Överlagring
- Exempel
- Inline-funktioner
- Exempel
- Exempel
- Exempel
- Klasser

Introduktion till C++

I detta kapitel ger vi en översikt över C++. Vissa delar har en fördjupning i andra kapitel i detta kursmateriel.

Eftersom C++ är en utvidgning av C innebär detta att nästan alla program som man kan skriva i C går att skriva exakt likadant i C++ (C är en delmängd av C++). Vi kommer att jämföra tilläggen i C++ med vanlig C för att belysa hur kraftfulla förbättringarna i C++ är.

När vi i detta kapitel ger en översikt över C++ kommer vi att visa huvudfunktionerna i C++. Våra exempel kommer i detta kapitel att vara relativt enkla för att enbart visa själva grundprincipen. Vi kommer senare att gå in på detaljer och regler. Detta kapitel är enbart till för att ge dig en kort översikt.

Historia och bakgrund

I början av 1980-talet började Bjarne Stroustrup vid AT&T:s Bell Laboratories i Murray Hill, New Jersey, att arbeta med en utvidgning av programspråket C som han kallade "C med klasser". Stroustrup är ursprungligen dansk och född i Århus.

Tillägget av klasser var det viktigaste tillägget jämfört med vanlig C, men även andra tillägg som kontroll av datatyper på funktionsargument gjordes. Klasser är det som gör att C++ kallas för ett objektorienterat programmeringsspråk. Den ursprungliga avsikten med programspråket var att göra programmerandet lättare för sina kollegor på Bell Laboratories, eftersom programmen tenderade att bli komplexa och enorma. "C med klasser" skulle vara kompatibelt med vanlig C samt införa klasser, vilket fanns i Simula 67.

Under 1983 omformades och utökades "C med klasser". Efter detta kallades språket för C++. De viktigaste utökningarna var virtuella funktioner och överlagring av operatorer.

1985 blev C++ tillgängligt utanför forskarkretsar och 1986 kom Bjarne Stroustrups bok "The C++ Programming Language" ut.

Utvecklingen av C++ fortsätter ännu och två nya tillägg till C++ kommer att bli templates och exception.

Standarder

Det pågår för närvarande arbete på att formellt standardisera C++ inom American National Standards Institute (ANSI). Den kommitté som har hand om detta heter X3J16.

C++ i relation till C och ANSI-C

C++ är baserat på C, vilket innebär att alla program som är skrivna i C går att använda på en kompilator för C++. Om du redan kan programmera i C behärskar du redan den grundläggande syntaxen i C++. Avsikten med C++ är att det ska vara kompatibelt med C men erbjuda fler möjligheter. För att kunna utnyttja C++:s nya möjligheter fullt ut bör man lära sig ett delvis nytt sätt att tänka för att kunna utnyttja exempelvis klasser på bästa tänkbara sätt.

Även om C och C++ utvecklas parallellt så påverkar de varandra klart märkbart. ANSI-C har anammat C++:s datatypskontroll. Denna möjliggör en kontroll av argumentens datatyper samt överlagring av funktioner. ANSI-C har även övertagit C++:s const. I C++ har man följt ANSI-standarden vad gäller vanlig C efter att den infördes.

I C++ uppmanas programmeraren att använda inline-funktioner hellre än preprocessorns #define-makron. Dessa inline-funktioner kompileras som makron. Inline-funktioner gör att funktioner betraktas som makron, vilket möjliggör kontroll av datatyper. Även användandet av konstanta värden med hjälp av const är till för att minska användandet av preprocessorn.

In- och utmatning är förbättrad i C++ tack vare streams. I C++ finns enradskommentarer.

Istället för att använda **malloc** och **free** för att kontrollera den dynamiska minneshanteringen som man gör i C bör man i C++ använda new och delete.

Men det största tillägget i C++ är införandet av klasser. Med hjälp av klasser kan man skapa egna datatyper. Klasser är avsevärt mer än strukturer i och med att man i klasser kan ha både funktioner och datatyper som medlemmar i en klass. Man kan definiera en typ av dataobjekt och de funktioner som kan utföras på den. Klasser är det som gör C++ till ett objektorienterat programspråk.

C++ är ett bättre C.

Reserverade ord

Eftersom C++ är en utvidgning av C så innebär detta att alla nyckelord som är reserverade i C (inklusive ANSI-C) är reserverade i C++.

Nedan följer en förteckning över de ord som är reserverade i C++:

asm	C++ och vissa implementeringar av C
auto	ANSI-C
break	ANSI-C
case	ANSI-C
catch	C++
char	ANSI-C
class	C++
const	ANSI-C
continue	ANSI-C
default	ANSI-C
delete	C++
do	ANSI-C
double	ANSI-C
else	ANSI-C
enum	ANSI-C
extern	ANSI-C
float	ANSI-C
for	ANSI-C
friend	C++
goto	ANSI-C
if	ANSI-C
inline	C++
int	ANSI-C
long	ANSI-C
new	C++
operator	C++
private	C++
protected	C++
public	C++
register	ANSI-C
return	ANSI-C
short	ANSI-C
signed	ANSI-C
sizeof	ANSI-C
static	ANSI-C
struct	ANSI-C
switch	ANSI-C
template	C++
this	C++
throw	C++
try	C++
typedef	ANSI-C
union	ANSI-C
unsigned	ANSI-C
virtual	C++
void	ANSI-C

```
volatile  ANSI-C
while     ANSI-C
```

Operatorer i C++

Nedan följer en förteckning över de operatorer som finns i C++:

```
::          C++
[]          ANSI-C
()          ANSI-C
->          ANSI-C
->*         C++
.           ANSI-C
.*          C++
!           ANSI-C
~           ANSI-C
++          ANSI-C
--          ANSI-C
unärt +         ANSI-C
unärt -         ANSI-C
unärt *         ANSI-C
unärt &         ANSI-C
sizeof      ANSI-C
new         C++
delete      C++
*           ANSI-C
/           ANSI-C
%           ANSI-C
+           ANSI-C
-           ANSI-C
<<          ANSI-C
>>          ANSI-C
<           ANSI-C
<=          ANSI-C
>           ANSI-C
>=          ANSI-C
==          ANSI-C
!=          ANSI-C
&           ANSI-C
^           ANSI-C
|           ANSI-C
&&          ANSI-C
||          ANSI-C
```

```
?:          ANSI-C
=           ANSI-C
+=          ANSI-C
-=          ANSI-C
*=          ANSI-C
|=          ANSI-C
&=          ANSI-C
&=          ANSI-C
^=          ANSI-C
<<=         ANSI-C
>>=         ANSI-C
,           ANSI-C
```

Kommentarer i C++

Det finns två sätt att skriva kommentarer. Dels kan man använda balanserade kommenterar (samma som i C) som kan sträcka sig över flera rader och dels kan man använda enradskommentarer (//). Kommentarer plockas bort av preprocessorn och är aldrig synliga för kompilatorn.

Exempel

// Detta är en enradskommentar

/* Detta är en balanserad kommentar
som kan sträcka sig över flera rader
Det anses vara god stil att använda
kommentarer */

Deklarationer i C++

En iögonfallande skillnad mellan vanlig C och C++ är variabeldeklarationen. I vanlig C måste alla deklarationer ske längst upp i en funktion. Deklarationerna ligger längst upp, innan de programinstruktioner som exekveras i programmet.

I C++ kan en deklaration placeras precis var som helst i programmet. Detta gör att man kan deklarera variablerna först när man behöver dem.

Exempel

XXXXXXXXXXXXXXXXXXXX

Även vad gäller deklaration av funktioners variabeltyper skiljer sig C och C++. Detta kommer vi att se närmare på längre fram i denna kursdokumentation.

In- och utmatning i C++ (streams)

I C++ hanteras in- och utmatning av data till och från enheter med hjälp av biblioteket streams. Om du vill använda detta bibliotek måste du inkludera filen **iostream.h** .

cin,cout och cerr

Om du inkluderat iostream.h har du tillgång till tre globala objekt för enkel in och utmatning till och från bildskärm och tangentbord dessa är;

```
*   cin     (standard input = inmatningsenheten, dvs
            tangentbordet)

*   cout    (standard output = utmatningsenheten, dvs
             bildskärmen)

*   cerr    (standard error = felenheten, dvs bildskärmen)
```

Exempel med cout

I nedanstående enkla program skriver vi ut en sträng på skärmen.

```
//streams1.cpp
#include <iostream.h>
main()
{

	cout << ”Hello World!\n”;
	return 0;
}
```

Exempel med cin

Om vi vill låta användaren mata in data kan vi kombinera objektet **cin** med >>-operatorn. Vi gör på samma sätt oberoende av vilken datatyp som vi vill tilldela värden:

```
// streams2.cpp
#include <iostream.h>
main()
{
	char fnamn[30];
	char enamn[30];

	cout << ”Ange förnamn:”;
	cin >> fnamn;

	cout << ”Ange efternamn:”;
	cin >> enamn;

	cout << ”Du heter tydligen ” << fnamn << ”  ”  << enamn << endl;

	return 0;
}
```

Exempel med cerr

Cerr motsvarar enheten för felmeddelanden. Om programmet vill meddela om fel bör detta ske mot detta objekt.

```
// streams3.cpp
#include <iostream.h>
#include <stdlib.h>  // för exit
main()
{
     double tal1;
     double tal2;

     cout << "Ange täljare :";
     cin >> tal1;

     cout << "Ange nämnare :";
     cin >> tal2;

     if (tal2 == 0)
     {
          cerr << "Du kan inte dividera med noll! Programmet avbryts!\n";
          exit(1);
     }

     cout << tal1 << "/" << tal2 << "  = " << tal1/tal2 << endl;

     return 0;
}
```

Dynamisk minneshantering i C++

I C++ hanteras dynamisk minneshantering med operatorerna **new** och **delete**. New och delete är unära operatorer. Genom att använda new kan man allokera minnesutrymme för objekt som har en av programmeraren begränsad livslängd. Objektets minnesutrymme frigörs sedan med operatorn delete.

```
// new1.cpp
#include <iostream.h>
#include <string.h>

int
main(int,char**)
{
     // Enstaka element...
     char * chptr  =  0;                  // En pekare till char

     chptr =  new char;                   // Allokera plats för en char...

     if (chptr != 0)                      // Om det gick bra....
     {
          *chptr = 'A';                   // Placera dit ett 'A'....
          cout << "chptr pekar på ett "   // Berätta....
                    << *chptr << endl;
          delete chptr;                   // Frigör heapen...
          chptr = 0;                      // Nollställ pekaren
     }
     else
     {
          cerr << "Kunde inte allokera plats för en char!\n";
     }

     // Fält ...
     char   text[] =  "Denna text skall kopieras!";
     char * kopia  =  0;

     kopia = new char[strlen(text)+1];   // Allokera plats för en kopia
                                         // av text...

     if (kopia != 0)                     // Om det gick bra....
     {
          strcpy(kopia,text);            // Kopiera ....

          cout << "Text :" << text  << endl; // Berätta...
          cout << "Kopia:" << kopia << endl;

          delete [] kopia;               // Frigör heapen från fältet..
          kopia = 0;
     }
     else
     {
          cerr << "Kunde inte allokera plats för ett teckenfält!\n";
     }

     return 0;

}
```

Felhantering

new returnerar värdet 0 (NULL) om det inte finns plats för det objekt man begärt. Man kan därför införa felhantering genom att testa returvärdet.

```
if (( text = new char[100]) == 0) {
   cerr << "Slut på minne!";
   exit(1);
}
```

Newhandler

Vi kan också införa felhantering genom att anropa funktionen set_new_handler på nedanstående sätt. I funktionen, vars namn anges som argument, kan vi själva ta hand om och eventuellt åtgärda fel som uppstår vid användning av **new**.
set_new_handler är deklarerad i filen **new.h**.

```
// new2.cpp
#include <iostream.h>
#include <stdlib.h>
#include <new.h>
#include <time.h>

int& getmax(int arr[],int size);
void myhandler(void);

int
main(int,char**)
{
     srand((unsigned) time(NULL));

     void (*old)() = set_new_handler(myhandler);
     int size = 0;
     for(int n = 0;n< 10;n++)
     {
          size = rand();
          cout << "Allokerar " << size << " int på heapen!\n";
          int * iptr = new int[size];
          for(int i=0;i<size;i++)
          {
               iptr[i] = rand();
          }
          cout << "Största värde " << getmax(iptr,size) << endl;
          cout << "Frigör " << size << " int på heapen!\n";
          delete [] iptr;
     }
     set_new_handler(old);

     return 0;
}

void myhandler(void)
{
     cerr << "Minnesbrist!\n";
     exit(1);
}

int& getmax(int arr[],int size)
{
     int* iptr = &arr[0];       // Antag att det första elementet är störst
     for(int i=1;i<size;i++)         // Iterera över hela fältet...
     {
          if (*iptr < arr[i])   // Om större..
          {
               iptr  = &arr[i];// Sätt till detta värde...
          }
     }
     return *iptr;
```

```
}
```

delete

Med **delete** operatorn lämnar vi tillbaka det minne som vi använt. Som operand måste vi använda ett pekarvärde som har returnerats av **new**.

```
char  *text    = new char[100];
long  *lfield  = new long[50];
float *ffaelt  = new float[100];
//.....
delete  [] text;
delete  [] lfield;
delete  [] ffaelt;
```

Referensvariabler

Referensvariabler är ett slags "alias" för variabler eller objekt. En referensvariabel måste alltid initieras.

```
// refer1.cpp
#include <iostream.h>

int
main(int,char**)
{
   int    value      = 200;
   int&   valueref   = value;

   cout << "Först.....\n";
   cout << "valueref:"   << valueref << endl;
   cout << "value   :"   << value   << endl;

   valueref++;

   cout << "Efter valueref++\n";
   cout << "valueref:"   << valueref << endl;
   cout << "value   :"   << value   << endl;

   value--;

   cout << "Efter value--\n";
   cout << "valueref:"   << valueref << endl;
   cout << "value   :"   << value   << endl;
```

```
    return 0;
}
```

Exempel

Genom att använda referensvariabler som parametertyp i en funktion kan funktionen manipulera med en aktuell parameter. Nedan blir variabeln **valueref** referens till variabeln **max**.

```
// refer2.cpp
#include <iostream.h>

void getmax(int& valueref, int arr[],int size)
{
       valueref = arr[0];         // Antag att det första elementet är störst
       for(int i=1;i<size;i++)    // Iterera över hela fältet...
       {
              if (valueref < arr[i])     // Om större..
              {
                     valueref = arr[i];  // Sätt till detta värde...
              }
       }
}


int
main(int,char**)
{
       int arr[]     = { 0,-6,-14,4,23,12,7,19,2 };
       int size      = sizeof(arr)/sizeof(arr[0]);
       int max       = 0;

       getmax(max,arr,size);

       cout << "Största värde är " << max << endl;

       return 0;

}
```

Referens som returvärde

Genom att låta en funktion returnera en referens kan vi initiera en referensvariabel med returvärdet från en funktion.

```
// refer3.cpp
#include <iostream.h>

int& getmax(int arr[],int size)
{
     int* iptr = &arr[0];  // Antag att det första elementet är störst
     for(int i=1;i<size;i++)    // Iterera över hela fältet...
     {
```

```
          if (*iptr < arr[i])   // Om större..
          {
               iptr  = &arr[i];// Sätt till detta värde...
          }
     }

     return *iptr;
}


int
main(int,char**)
{
     int arr[] = { 5,6,14,4,23,12,7,19,2};
     int size = sizeof(arr)/sizeof(arr[0]);

     for(int i=0;i<=size;i++)
     {
          int& maxref = getmax(arr,size);
          cout << "Största värde är " << maxref << endl;
          maxref = 0;
     }

     return 0;
}
```

Överlagring

Med hjälp av överlagring av funktioner kan man i C++ ha samma namn på två eller flera funktioner. Varje funktion måste dock ha minst ett argument med en datatyp som särskiljer den från de andra funktionerna med samma namn. Kompilatorn särskiljer mellan de funktioner som har samma genom att kontrollera antalet argument och datatypen på argumenten. Med hjälp av datatypen på argumenten vid anropet förstår kompilatorn automatiskt vilken funktion som anropas.

Exempel

I detta exempel deklareras tre överlagrade funktioner med namnet input samt tre överlagrade funktioner med namnet print. Kompilatorn väljer den funktion som matchar den parametertyp som används vid respektive anrop.

```
// overlag1.cpp
#include <iostream.h>

void input(int& value);
void input(char& ch);
void input(double& d);
void print(int value);
void print(char ch);
void print(double d);

int
main(int,char**)
{
   int         i;
   char    ch;
   double   d;

   input(i);
   input(ch);
   input(d);

   print(i);
   print(ch);
   print(d);

   return 0;
}
void input(int& value)
{
   cout << "Ange ett heltalsvärde:";
   cin >> value;
}
void input(char& ch)
{
   cout << "Ange ett tecken:";
   cin >> ch;
}
void input(double& d)
{
   cout << "Ange ett flyttalsvärde:";
```

```
   cin >> d;
}
void print(int value)
{
   cout << "Heltalsvärde:" <<  value << endl;
}
void print(char ch)
{
   cout << "Tecken :" << ch << endl;
};
void print(double d)
{
   cout << "Flyttalsvärde:" << d << endl;
}
```

Inline-funktioner

Vanliga makron i C har några svagheter, som till exempel att de inte kan hantera variabler och att man måste vara mycket noggrann när man konstruerar dem så att de inte expanderas på fel sätt. Visserligen kan man undvika expansionsfel genom att använda parenteser men fel på grund av fel datatyp i anropet finns det ingen kontroll av.

I C++ förbigår man dessa svagheter genom att använda inline-funktioner istället för makron. Modifieraren **inline** anger att en funktion ska inkluderas vid anropet och inte generera ett funktionsanrop.

Genom att deklarera funktioner som inline får vi i regel ett större men snabbare program än om vi inte använder inline.

Det är inte säkert att en kompilator klarar att implementera en funktion som inline. Vissa kompilatorer brukar påpeka om så är fallet. Det är vanligt att en kompilator inte klarar inline om funktionen innehåller en loop. Man kan även anta att det är svårt att implementera en rekursiv funktion som inline.

```
// inline1.cpp
#include <iostream.h>

inline int funk1(int value)
{
         return value * value;
}

inline int funk2(int a, int b)
{
         if (a>b)
              return a;
         else
```

```
                return b;
}

inline int funk3(signed char ch)
{
     int rc = 0;

     switch (ch)
     {
          case 'a': case  'A':
          case 'e': case  'E':
          case 'i': case  'I':
          case 'o': case  'O':
          case 'u': case  'U':
          case 'y': case  'Y':
          case 'ä': case  'Ä':
          case 'å': case 'Å':
          case 'ö': case 'Ö':
                                                  rc = 1;
                                                  break;
          default:
                                                  rc = 0;
                                                  break;
     }

     return rc;
}

inline void funk4(double delay)
{
     for(double i=0.0;i<delay; i+= 0.001);
}

inline void funk5(double delay)
{
     double i=0.0;
     while (i<delay)
     {
                i+= 0.001;
     }
}

inline void funk6(double delay)
{
     double i=0.0;
     do {
                i+= 0.001;
     }
     while (i<delay);
}


inline void funk7(const char * str)
{
     if (*str!='\0')
          funk7(++str);
     else
          return;
     cout << *--str;
}
```

```
int
main(int,char**)
{
          cout << "funk1(10)  " << funk1(10)    << endl;

          cout << "funk2(1,2) " << funk2(1,2)  << endl;

          cout << "funk3('Ö') " << funk3('Ö') << endl;

          cout << "funk4(1000)" << endl;
          funk4(1000);

          cout << "funk5(1000)" << endl;
          funk5(1000);

          cout << "funk6(1000)" << endl;
          funk6(1000);

          funk7("Test av inline med rekursiv funktion!");

          return 0;
}
```

Exempel

I nedanstående exempel utvärderar vi om det går fortare att använda en inlinefunktion än en vanlig funktion

```
// inline2.cpp
#include <iostream.h>
#include <stdlib.h>
#include <time.h>
void swap(double& ref1, double& ref2)
{
     double temp = ref1;
     ref1 = ref2;
     ref2 = temp;
}

void sortera_swap(double array[], int size)
{
     const int outerstop = size - 1;
     const int innerstop = size;
     register int outerindex,innerindex;

     for(outerindex=0;outerindex < outerstop;outerindex++)
          for(innerindex=outerindex+1;innerindex < innerstop;innerindex++)
               if (array[outerindex] > array[innerindex])
                    swap(array[outerindex], array[innerindex]);
}

inline
void inlineswap(double& ref1, double& ref2)
{
     double temp = ref1;
     ref1 = ref2;
     ref2 = temp;
}


void sortera_inlineswap(double array[], int size)
{
     const int outerstop = size - 1;
     const int innerstop = size;
     register int outerindex,innerindex;

     for(outerindex=0;outerindex < outerstop;outerindex++)
          for(innerindex=outerindex+1;innerindex < innerstop;innerindex++)
               if (array[outerindex] > array[innerindex])
                    inlineswap(array[outerindex], array[innerindex]);
}

void slumpa(double array[], int size)
{
     for(int i=0;i<size;i++)
          array[i] = rand() + (double) rand()/(double) RAND_MAX;
}

int
main(int,char**)
{
     int rc = 0;                    // Returvärde från programmet...
```

```
    const int size = 3000;          // Lagom många för din miljö....
    double *array = new double[size]; // Använd heapen.....

    if (array!=0)
    {
         time_t start,stopp;        // Start- och stopptid för tidtagning

         srand(1);                  // Sätt slumptalsfrö
         slumpa(array,size);        // Generera tal....

         cout       << "Startar sortering av " << size
                    << " double med inlineswap\n";
         start = time(0);                // Ta starttid
         sortera_inlineswap(array,size);
         stopp = time(0);                // Ta stopptid
         cout  << "Inline swap " << size
               << " tal, tid:" << (stopp - start) << endl;

         srand(1);             // Sätt samma slumptalsfrö
         slumpa(array,size);   // Generera samma tal....
         cout       << "Startar sortering av " << size
                    << " double med vanlig swap\n";
         start = time(0);                // Ta starttid
         sortera_swap(array,size);
         stopp = time(0);           // Ta stopptid
         cout << "Swap utan inline " << size
                    << " tal, tid:" << (stopp - start) << endl;

         delete [] array;      // Frigör
         rc = 0;               // Allt gick bra
    }
    else
    {
         cerr << "Kan inte allokera plats för " << size << " double\n";
         rc = 1;    // Det gick inget vidare..
    }

    return rc;
}
```

Defaultinitierade variabler

I C++ är det möjligt att tillåta att en funktion kan anropas utan alla argument. De argument som utelämnats får förutbestämda värden. Det är endast de sista argumenten som kan få förutbestämda värden. Det går inte att låta det första argumentet få ett förvalt värde och sedan låta det andra och det tredje argumentet få sina värden via funktionsanropet.

Exempel

Funktionen **addera** kan anropas med noll, en, två, tre, fyra eller fem parametrar.

```
// default1.cpp
#include <iostream.h>

int addera(int a = 0, int b = 0, int c = 0,int d = 0,int
```

```
e=0);

main()
{
       cout << addera() << endl;
       cout << addera(1) << endl;
       cout << addera(1,2) << endl;
       cout << addera(1,2,3) << endl;
       cout << addera(1,2,3,4) << endl;
       cout << addera(1,2,3,4,5) << endl;

       return 0;
}

int addera(int a, int b, int c,int d,int e)
{
   return a + b + c + d + e;
}
```

Exempel

Funktionen **multiplicera** kan anropas med en, två, tre, fyra eller fem parametrar. Den första parametern måste alltid få ett värde via anropet men de andra kan utelämnas.

```
// default2.cpp
#include <iostream.h>

int multiplicera(int a, int b = 1, int c = 1,int d =
1,int e = 1);

main()
{
         cout << multiplicera(1) << endl;
         cout << multiplicera(1,2) << endl;
         cout << multiplicera(1,2,3) << endl;
         cout << multiplicera(1,2,3,4) << endl;
         cout << multiplicera(1,2,3,4,5) << endl;

         return 0;
}

int multiplicera(int a, int b, int c,int d,int e)
{
     return a * b * c * d * e;
}
```

Klasser

Den stora fördelen i C++ jämfört med vanlig C är införandet av klassbegreppet . Klasser är en utvidgning av begreppet struct i vanlig C. Med hjälp av klasser kan man införa en datatyp som är definierad av användaren och funktioner och operatorer som är sammankopplade med klassen.

Klassen delas upp i delar som antingen är privata eller publika. Detta görs genom att använda nyckelorden private och public. De data som hör till den privata delen är endast tillgänglig för de funktioner som hör till klassen (samt de funktioner som är deklarerade som friend). Funktioner som hör till den privata delen kan endast anropas från andra funktioner som hör till samma klass

Nedan följer ett exempel på användning av klasser i C++:

Arv

I C++ kan man låta en klass ärva egenskaper från en annan klass. Klassen som ärver egenskaper brukar kallas för härledd klass eller subklass. Klassen som den ärver ifrån brukar kallas basklass eller superklass. Subklassen kan sedan vara basklass för en annan klass osv.

En hierarki av relaterade datatyper kan skapas som delar kod. De härledda klasserna ärver basklassens egenskaper samt får de egna specifika egenskaper som angetts vid kodningen av den deriverade klassen.