Programmering i C++
EDAF30
Typer

Typer

Innehåll

- Heltalstyper
- Flyttalstyper
- Pekare
- Minnesallokering
- Funktionspekare
- Typdeklarationer med `typedef`
- Typomvandlingar (casting)
- Uppräkningstyper
- Tabeller och par

Heltalstyper

- `signed char` (8 bitar) -127 ... 128
- `short int` (8 el. 16 bitar)
- `int` (16 el. 32 bitar)
- `long` (32 bitar)
- `unsigned char` (8 bitar) 0 ... 255
- `unsigned short` int (8 el.16 bitar)
- `unsigned int` (16 el. 32 bitar)
- `unsigned long` (32 bitar)

Heltalstyper

Test med sizeof i CodeBlocks

```
#include <iostream>
using namespace std;
int main () {
    cout << "sizeof(char)= " << sizeof(char)<<endl;
    cout << "sizeof(int) = " << sizeof(int) <<endl;
    cout << "sizeof(long)= " << sizeof(long)<<endl;
    cin >> ws; // Bryt med Ctrl-C ...
}
```

```
sizeof(char)= 1
sizeof(int) = 4
sizeof(long)= 4
```

Heltalstyper – Test av talområdet via casting

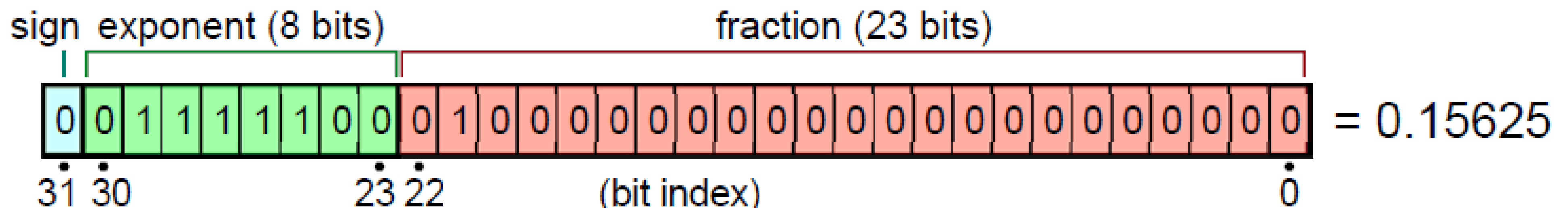
```
int main () {
 cout << "(char) -1 = " << (int)(char) -1 << endl;
 cout << "(unsigned char) -1 = " << (int)(unsigned char) -1 << endl;
 cout << "(short int) -1 = " << (short int) -1 << endl;
 cout << "(unsigned short int) -1 = "<<(unsigned short int)-1<<endl;
 cout << "(int) -1 = " << (int) -1 << endl;
 cout << "(unsigned int) -1 = " << (unsigned int) -1 << endl;
 cout << "(long) -1 = " << (long) -1 << endl;
 cout << "(unsigned long) -1 = " << (unsigned long) -1 << endl;
 cin >> ws;
}
```

```
(char) -1 = -1
(unsigned char) -1 = 255
(short int) -1 = -1
(unsigned short int) -1 = 65535
(int) -1 = -1
(unsigned int) -1 = 4294967295
(long) -1 = -1
(unsigned long) -1 = 4294967295
```

Flyttalstyper

- `float` (32 bitar)
- `double` (64 bitar)
- `long double` (96 bitar)

Typiskt för 32 bitars representation:
1 + 8 + 23 bitar (tecken + exponent + mantissa)



$$+1.01_2 \cdot 2^{124-127} = 1.25 \cdot 2^{-3}$$

Flyttalstyper

Test med sizeof i CodeBlocks

```
#include <iostream>
using namespace std;
int main () {
    cout << "sizeof(float)="
         << sizeof(float)<<endl;
    cout << "sizeof(double)="
         << sizeof(double) << endl;
    cout << "sizeof(long double)="
         << sizeof(long double) << endl;
    cin >> ws; // Bryt med Ctrl-C ...
}
```

```
sizeof(float)=4
sizeof(double)=8
sizeof(long double)=12
```

Pekare

Indirekt adressering av liknande typ som referenser i Java. Pekare kan dock manipuleras mera fritt (med de risker det medför)

Syntax

Asterisk före pekarvariabelnamn ger det som pekaren pekar på
(*p ger det som p pekar på)

Exempel

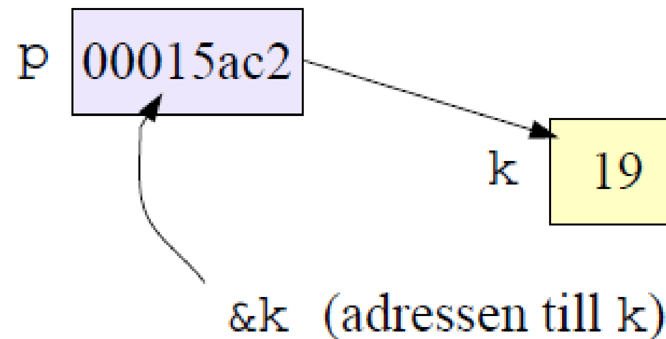
```
int *p;    // Betyder att p är pekare till int
int k=19;
p = &k;    // Nu har k fått ett namn till (*p)
*p = 17;   // Ändrar även k till 17
```

Pekare

Dereferensoperatorn *

Ger det som pekaren pekar på, dvs det vars adress ligger lagrad i pekarvariabeln

```
int *p;int k=19;
p = &k;
```



Referensoperatorn &

Ger adressen till en variabel

Pekare

Pekare till int

```
int *p1, *p2;
int n=4, m=5;

p1 = &n;    // p1 pekar på n
p2 = &m;    // p2 pekar på m
*p1 = 7;    // n blir 7
*p2 = *p1; // m blir också 7
p2 = p1;    // p2 pekar på n
```

Pekare på konstanter och konstant pekare

```
int k; // ändringsbar int
const int c = 100;// konstant int
const int *pc; // pekare till konstant int
int *pi; // pekare till ändringsbar int

pc = &c; // OK
pc = &k; // OK, men k kan inte ändras via *pc
pi = &c; // FEL! pi får ej peka på en konstant
*pc = 0; // FEL! pc pekar på en konstant

int *const cp = &k; // Konstant pekare
cp = 0; // FEL! Pekaren ej flyttbar
*cp = 123; // OK! Ändrar k till 123
```

Pekare

Sammanfattning

```
typ *p;     // p får typen
            // "pekare till typ"
p = &v;     // p tilldelas adressen till v

*p = 12;    // Det som p pekar på
            // tilldelas värdet 12
p1 = p2;    // p1 pekar på samma som p2
*p1 = *p2;  // Det som p1 pekar på
            // tilldelas samma värde
            // som det som p2 pekar på
```

Pekare och referenser

Referenser till int

```
int n=7, m, *p;
int& r1 = n;    // r1 är en referens till n
r1 = 7;         // n får värdet 7
int& r2 = m;    // Referenser måste initieras
r2 = r1;        // m får värdet 7
p = &r1;        // p pekar på n
*p = 4;         // n får värdet 4
r2 = *p + 1;    // m får värdet 5
```

Pekare och referenser

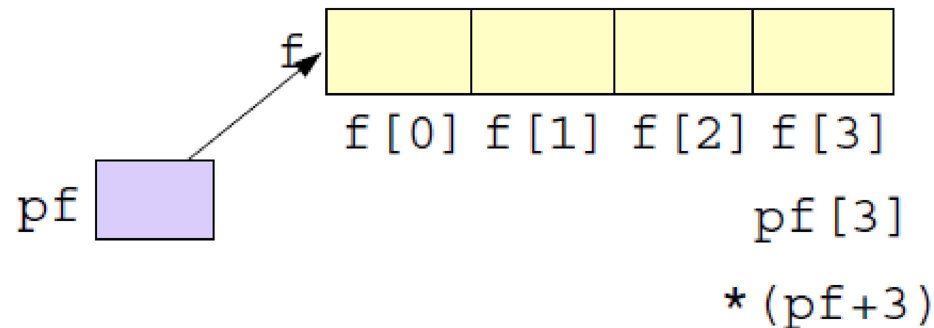
Pekarversion och referensversion av swap

```
// Pekarversionen
void swap(int *pa, int *pb) {
   int tmp = *pa;
   *pa = *pb; *pb = tmp;
}
// Referensversionen (som innan)
void swap(int& a, int& b) {
   int tmp = a;
   a = b; b = tmp;
}
...
int m=3, n=4;
swap(m,n);   // Referensversionen används
swap(&m,&n); // Pekarversionen används
```

Pekare och fält

Fält kan adresseras m.h.a. pekare

```
float f[4];          // 4 st float
float *pf;           // pekare till float
pf = f;              // samma som pf = &f[0]
float x = *(pf+3);  // Alt. x = pf[3];
x = pf[3];           // Alt. x = *(pf+3);
```



Pekare och fält

Nollställning av fältet f

```
// Alt. 1
for (int i=0; i<4; i++)
   f[i] = 0;

// Alt. 2
for (float *p=f; p<f+4; p++)
   *p = 0;
```

Nollställning av fältet f – C++11

```
// Alt. 3 (C++11)
for (float &e : f)
   e = 0;
```

Pekare och fält

Funktion för nollställning av heltalsfält

```
void zero(int *x, int n) {
   for (int *p=x; p < x+n; p++)
      *p= 0;
}
```

Fungerar eftersom endast *adressen* värdekopieras medan *innehållet* ändras

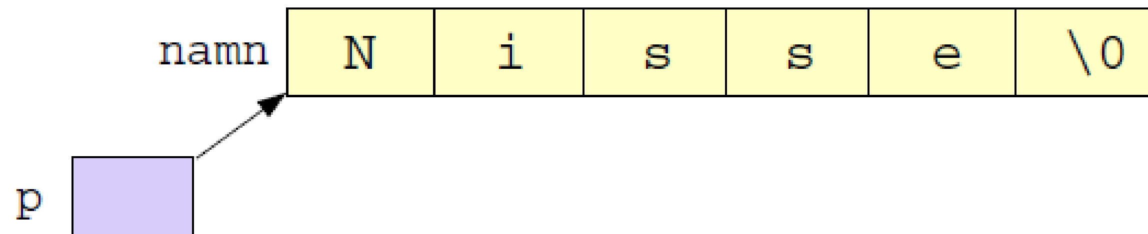
Indexering av heltalsfältet är oftast att föredra

```
void zero(int x[], int n) {
   for (int i=0; i < n; i++)
      x[i]= 0;
}
```

Pekare och textsträngar

Textsträngar i form av fält med char"

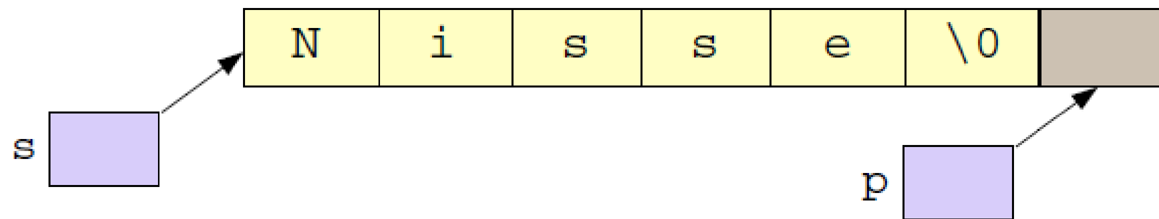
```
char namn[] = "Nisse";
char *p;
p = namn;
cout << p << endl;
cout << p+3 << endl;     // Skriver en delsträng
cout << *(p+3) << endl; // Skriver en char

Nisse
se
s
```



Pekare och textsträngar

Puzzle Corner version av `strlen` (ger stränglängden)

```
// Användning av const char* för
// att garanterat inte ändra något
int strleng(const char *s) {
    const char *p; // Pekaren kan ändras!
    for (p = s; *p++; )
        ;
    return p-s-1;
}
```



Pekare och textsträngar

Exempel på fält av pekare (ett datums veckodag)

```
char* wdays[] = {"måndag", "tisdag", "onsdag",
         "torsdag", "fredag", "lördag", "söndag"};
int main(int argc, char* argv[]) {
   int y, m, d, ind;
   if (argc!=4) {
      cout <<"Anropa enligt mönstret " << argv[0]
           << "yyyy mm dd" << endl;
      return -1;
   }
   y=atoi(argv[1]); m=atoi(argv[2]); d=atoi(argv[3]);
   if (m<3) {m+=12; y--;} // Zeller's kongruens (1886)!
   ind = (d + 2*m + (3*(m+1))/5
         + y + y/4 - y/100 + y/400) % 7;
   cout << argv[1] << "-" << argv[2] << "-"
        << argv[3] << " är en " << wdays[ind] << endl;
} // Datumformat enligt standarden ISO8601
```

Minnesallokering

Utrymme för dynamiska variabler skapas med `new`

```
double *pd = new double;
*pd = 3.141592654;
float *px, *py;
px = new float[20];
py = new float[20];
```

Frigörande av minne görs med `delete`

```
delete pd;
delete [] px; // [] krävs för fält
delete [] py;
```

Minnesallokering

Typiskt misstag: Att glömma allokera minne

```
char namn[80];
char *p;      //Ingen varning vid kompilering
char str[];   // Kompileringsfel med Gnu!
*namn = 'Z'; // Ok. Ger namn[0]='Z'
*p = 'Z';     // Fel! 'Z' hamnar i icke
              // allokerat utrymme(krasch)
strcpy(p, "Nisse Nilsson");
// Ger garanterat exekveringsfel
// ("Segmentation fault" i Linux)
```

Minnesallokering

Fler misstag: En misslyckad read_line

```
char *read_line() {
   char temp[80];
   cin.getline(temp, 80);
   return temp; // Lokalt statiskt reserverat
                // utrymme deallokeras vid retur!!!
}
char namn[80];
int main () {
   cout << "Ange ditt namn: ";
   strcpy(namn, read_line());
   cout << "Goddag " << namn << endl;
}
```

Minnesallokering

Tydligare exempel av misslyckad read_line

```
char *read_line() {
   char temp[80];
   cin.getline(temp, 80);
   return temp; // Lokalt statiskt reserverat
                 // utrymme deallokeras vid retur!!!
}
int main () {
    char *namn, *ort;
    cout << "Ange ditt namn: ";
    namn = read_line();
    cout << "Ange din bostadsort: ";
    ort = read_line();
    cout << "Goddag " << namn << " från " << ort << endl;
}
```

Minnesallokering

Delvis korrigerad version av read_line

```
char *read_line() {
   char temp[80];
   cin.getline(temp, 80);
   char *res = new char[strlen(temp)+1];
   strcpy(res, temp);
   return res; // Dynamiskt allokerat överlever
}
char namn[80];
int main () {
   cout << "Ange ditt namn: ";
   strcpy(namn, read_line());  // men minnesläcka här!
   cout << "Goddag " << namn << endl;
}
```

Minnesallokering

Ytterligare korrigerad version av read_line

```
char *read_line() {
    char temp[80];
    cin.getline(temp, 80);
    char *res = new char[strlen(temp)+1];
    strcpy(res, temp);
    return res; // Dynamiskt allokerat överlever
}
int main () {
    char *namn, *ort;
    cout << "Ange ditt namn: ";
    namn = read_line();  // Ta över ägarskap av sträng
    cout << "Ange din bostadsort: ";
    ort = read_line();
    cout << "Goddag " << namn << " från " << ort << endl;
    delete [] namn;     // Avallokera sträng
    delete [] ort;
}
```

Funktionspekare

Pekare kan också peka på funktioner

```
float (*pf)(int,int);
float hypotenusa(int a, int b) {
   return sqrt((float)(a*a + b*b));
}
float medelv(int x, int y) {
   return ((float)(x + y))/2;
}
int main () {
   pf = hypotenusa; cout << pf(3,4) << endl;
   pf = medelv; cout << pf(3,4) << endl;
}
```

Funktionspekare

Funktioner kan vara argument till funktioner

```
float hypotenusa(int a, int b) {
    return sqrt((float)(a*a + b*b));
}
float medelv(int x, int y) {
    return ((float)(x + y))/2;
}
float eval(float (*f)(int,int), int m, int n) {
    return f(m, n);
}
int main () {
    cout << eval(hypotenusa, 3, 4) << endl;
    cout << eval(medelv, 3, 4) << endl;
}
```

Typdeklarationer med typedef

Nya typer kan deklareras med typedef

```
typedef unsigned long ulong;
typedef char rad[80];
typedef double (*funpek)(int);

// Följande deklarationer
ulong m, n, a[12];
rad namn;
funpek pf;

// svarar då mot deklarationerna
unsigned long m, n, a[12];
char namn[80];
double (*pf)(int);
```

Typomvandlingar (casting)

Automatiska typomvandlingar

- Uttryck av typen $x \odot y$ där $\odot$ är en binär operator
  Ex: `double + int ==> double`
  `float + long + char ==> float`
- Tilldelningar och initieringar: Värdet i högerledet konverteras till samma datatyp som i vänsterledet
- Konvertering av typen hos aktuella parametrar till typen för de formella parametrarna
- Fält ==> pekare
- 0 ==> `NULL` (tom pekare, `null_ptr` i C++11)

Typomvandlingar (casting)

Explicita typomvandlingar

Syntax

```
(typnamn)uttryck; // (Både i C och i C++)
typnamn(uttryck); // (Endast i C++)
```

Exempel

```
char *p = (char *) 07650; // heltal ==> pekare
long i = (long) p;        // pekare ==> heltal
int x=3, y=4;
double a = ((double)(x+y))/2;
cout << (int)(unsigned char) -3 << endl;
cout << (int)(char) 1000 << endl;
```

Uppräkningstyper

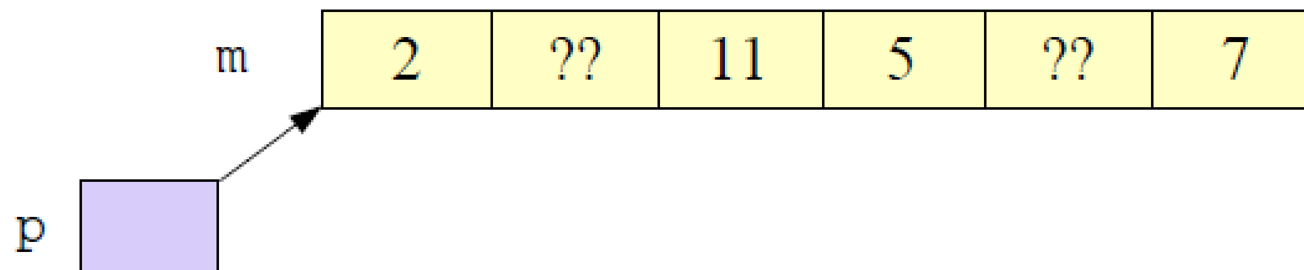
Möjlighet att namnge värden m.h.a. enum

```
enum color {blue, red, green, white};
enum ans {ja, nej, kanske, vetej};

// Deklaration av variabler
color fgcol=blue, bgcol=white;
ans svar;

// Tilldelningar
fgcol=red; bgcol=green;
svar = nej;
fgcol = kanske; // Fel!
svar = 2; // Fel!
```

Tabeller

Flerdimensionella fält (t.ex. matriser)

```
int m[2][3]; // En 2x3-matris

m[1][2] = 7;
m[0][0] = 4;
m[1][0] = 5;
int *p;
p = m;         // Fungerar inte!
p = &m[0][0];
*p = 2;
p[2] = 11;
```



Tabeller

Parametrar av typ flerdimensionella fält

```
void addone(int a[][3]) {
   for (int i=0;i<2;i++) {
      for (int j=0;j<3;j++) {
         a[i][j]++;
      }
   }
}
void printmatr(int a[][3]) {
   for (int i=0;i<2;i++) {
      for (int j=0;j<3;j++) {
         cout << setw(8) << a[i][j] << " ";
      }
      cout << endl;
   }
}
```

Par

Enkelt sätt att definiera par av olika datatyper

```
#include <utility>
...
pair<string, int> p1("Nils", 42), *pp;
cout << p1.first << " "
     << p1.second << endl;
pp = &p1;
pp->second = 43; // pp-> <==> (*pp).
pp->first = "Nisse";
cout << p1.first << " "
     << p1.second << endl;
```