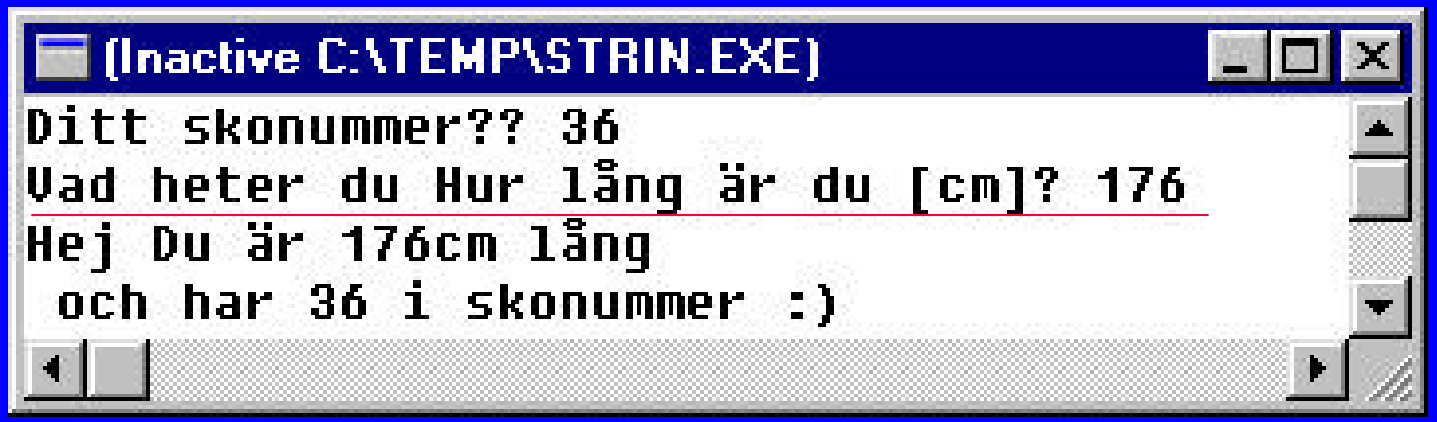
Arrayer och pekare
Något om operatorer
… men först...

Blandad inmatning

```
#include<iostream.h>
void main(void){
   char fnamn[20]; float lengd; int skonummer;
   cout << "Ditt skonummer?? "; cin >> skonummer;
   cout << "Vad heter du ";
   cin.getline(fnamn,20);
   // Skriv ut resultatet
   cout << "Hur lång är du [cm]? "; cin >> lengd;
   cout << "Hej " << fnamn
        << "Du är " << lengd
       << "cm lång\n och har "
        << skonummer << " i skonummer :)";
}
```

Blandad inmatning

Resultat av en körning



Vart tog namnet vägen?

Blandad inmatning

```
cin >> skonummer;
```

- Läser in teckensträngen "36" och omvandlar den till talet 36.
- Inläsningen lämnar kvar radlutstecknet i inmatningsbuffert
- **`cin.getline()`** träffar på radslut och avbryter inmatningen.
- Strängen **`fnamn`** blir tom, dvs **`""`**

Töm inbuffern

Lösning:

```
cin >> skonummer;
cin.get(); // Läser ett tecken
cin.getline(fnamn,20);
cin >> lengd;
```

Sammanfattning

```
char bytes[20]; // Array of char
```

Minne reserveras (*allokeras*) för 20 bytes

```
int heltal[40];  // plats för 40 heltal
float flista[8]; // plats för 8 flyttal
```

Åtkomst av element:

```
bytes[5] = 'A'; // Obs! - ’enkelfnutt'
cin >> flista[0] >> flista[1];
```

Sammanfattning

Strängar

```
char inrad[81];
```

Rymmer 80 skrivbara tecken 0 - 79, och '\0'

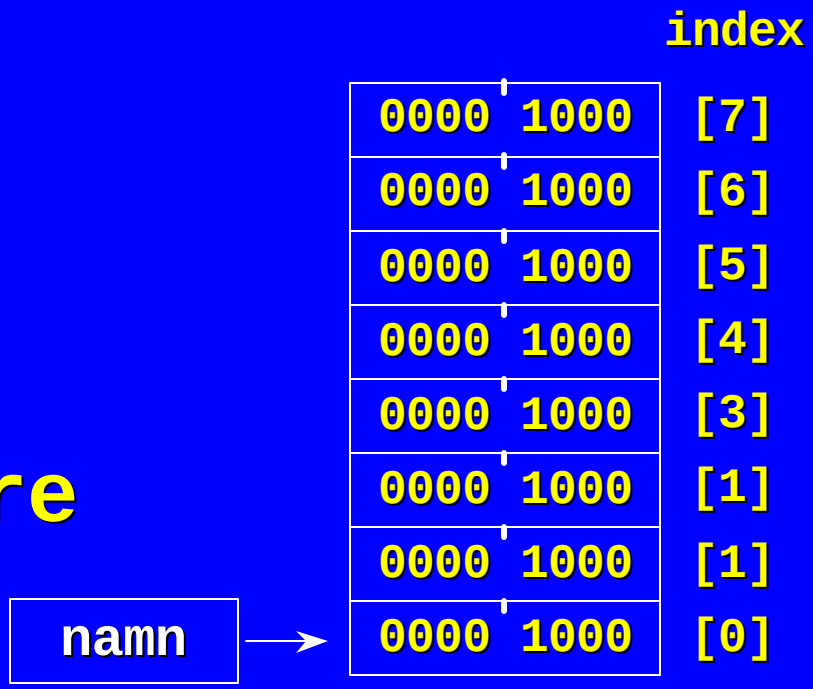
Se upp med:

- Glöm inte null vid tilldelning ett tecken i taget!
- Spärra inmatning av för många tecken!
- Töm inmatningsbuffern före användning av **`getline()`**

Arrayer och pekare

```
// Deklaration av en array av char
char namn[8];
```

namn är en pekare



Pekarvariabler

En pekarvariabel
- innehåller en minnesadress
- pekar på ett dataobjekt av viss typ

```
// Deklaration:
char *pek; // Pekare till char
```

Variabelns namn är **`pek`** (`inte *pek`)

`pek` måste ges ett värde

Pekarvariabler

Värdet av en pekarvariabel är en minnesadress.

Pekare deklareras att peka på dataobjekt av viss typ

```
char  *cpek; // kan peka på tecken
int   *ipek; // kan peka på heltal
float *fpek; // kan peka på flyttal
```

Hur får pekaren sitt värde?

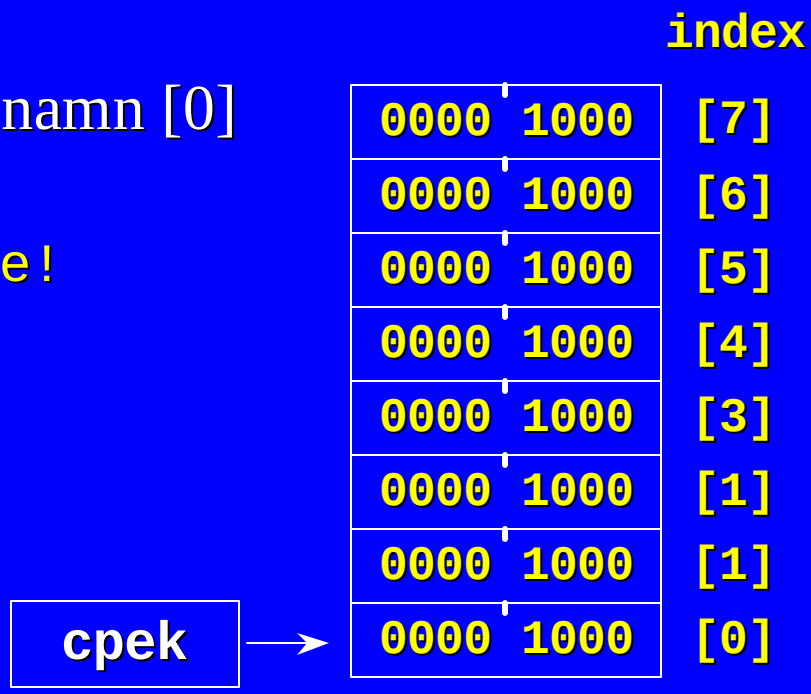
Pekarens värde

Deklaration av en array av **char**

```
char namn[8]; // deklaration av en teckenvektor
char *cpek;   // deklaration av pekare till char
```

Låt pekaren peka på elementet namn [0]

```
cpek = namn;
// namn är redan en pekare!
```



Pekarens värde

Initiering till viss minnesadress

```
int *minne1 = 0xFF;
```

Ovanligt - men kan användas då man vill peka ut en bestämd minnesposition

Adressoperatorn &

Operatorn & [*et*] tar fram
minnesadressen till en variabel

```
// Deklaration av variabler
int antal;
float volym;
// Deklaration av pekare
int *antpek;   // odefinierade..
float *volpek; // pekare inte på någonting
// Låt pekarna peka på antal och volym
antpek = &antal; // pekar på antal
volpek = &volym; // pekar på volym
```

Adressoperatorn &

Skillnad mellan enkla variabeltyper och array:

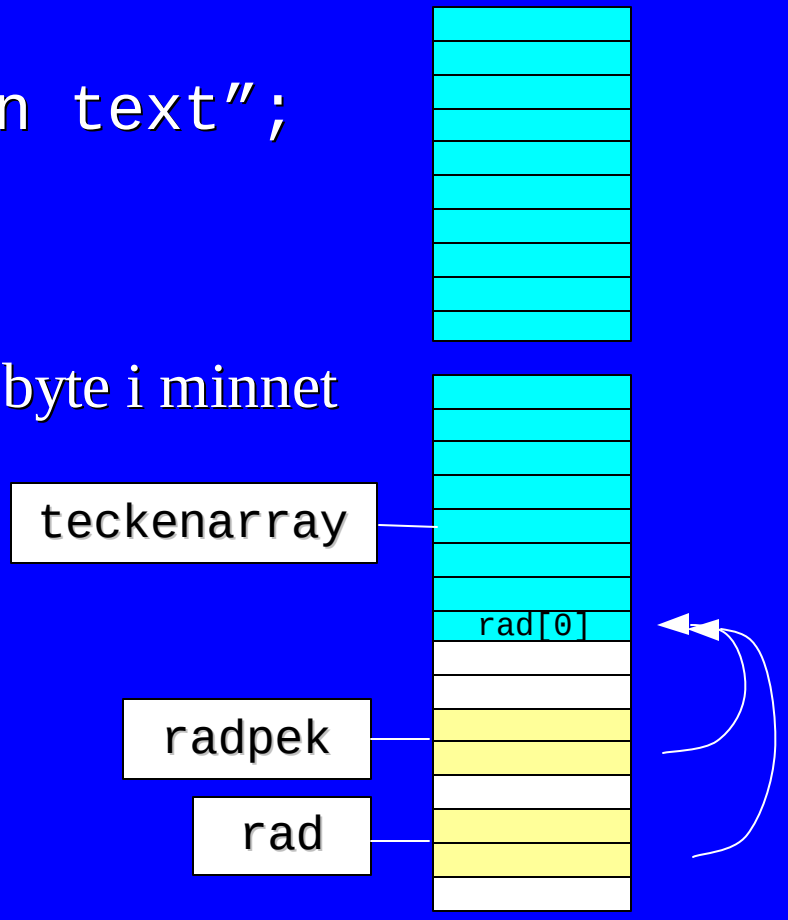
```
char rad[81]; // rad är en pekare
char *radpek = rad;

float volym = 200; // volym är en variabel
// Adressen till volym är &volym
float *volpek = &volym;
```

Peka på en array

```
char rad[] = ”Detta är en text”;
char *radpek = rad;
```

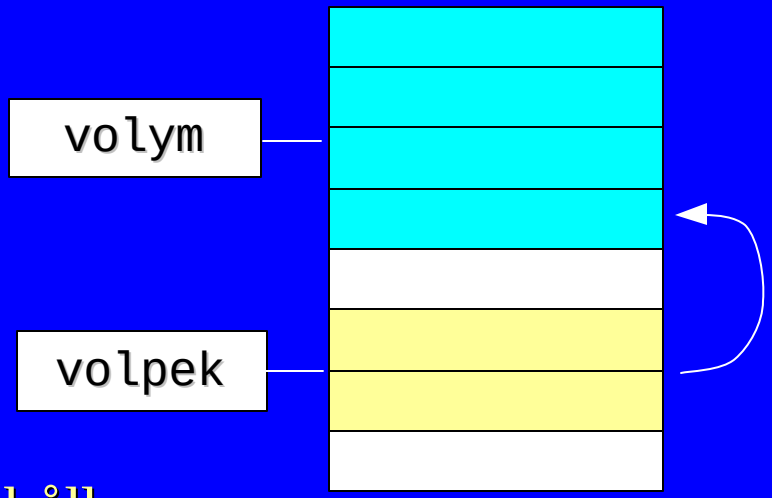
radpek och **rad** pekar på samma byte i minnet

Nämligen på `rad[0]`



Peka på en variabel

```
float volym = 200; // volym är en variabel
// Adressen till volym är &volym
float *volpek = &volym;
```



En pekare är en variabel som innehåller
en adress till ett dataobjekt i minnet

Att komma åt värde via pekare

Åtkomst med operatorn *

```
float a, b, *apek; // ev på samma rad
a = 450;
apek = &a; // apek pekar på a
// Hämta värdet med pekaren
b = *apek;
```

Detta kallas att *avreferera* pekaren

Många stjärnor

Vid multiplikation *(multiplikationsoperator)*

```
volym = bottenyta * hojd;
```

Vid deklaration av pekare

```
double *dpek; // markör - inte en operator
```

Vid avreferering av pekare (*derferensoperator*)

```
double q;
q = *dpek; // q får det värde dpek pekar på
```

Varför har pekare en typ?

En (vanlig) pekare tar alltid lika stor plats i minnet.

```
float *f; // pekar på en float
int   *i; // pekar på en int
char  *c; // pekar på en char
```

dvs pekaren ”vet” hur stort minnesutrymme den pekar på.

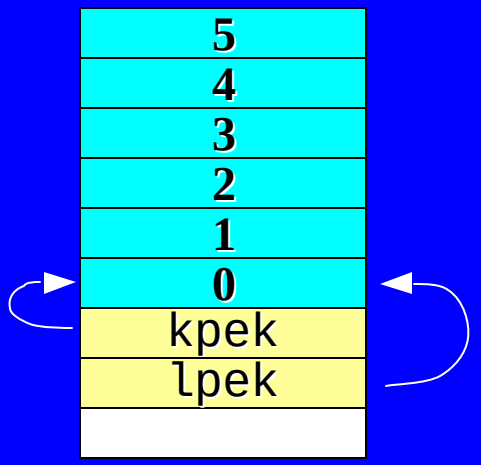
Varför?

- Svaret ges vid pekarstegning , dvs aritmetik på pekare.

Pekararitmetik

En pekare kan ges värdet av en annan pekare:

```
int lista[] = {0,1,2,3,4,5};
int *lpek, *kpek;
lpek = lista; // lpek pekar på lista[0]
kpek = lpek;  // kpek pekar på lista[0]
```



Pekararitmetik

Tillåten (och rimlig) aritmetik på pekare

```
int lista[] = {0,1,2,3,4,5}, *lpek;
lpek = lista;

lpek = lpek + 1; // stega upp två bytes
lpek++;          // stega upp två bytes
lpek+=2;         // stega upp fyra bytes
lpek--;          // stega ner två bytes
lpek-=4;         // stega ner åtta bytes
```

Varje pekarsteg motsvarar datatypens minnesutrymme!

Pekarstegning vs indexering

```
#include <iostream.h>
void main(void){
  int lista[] = {0,1,2,3,4,5};
  int *lpek = lista; // peka på element med index = 0
   // Skriv ut värden med indexering
   for (int i = 0; i < 6; i++)
       cout << lista[i] << endl;
   // Skriv ut samma värden med pekarstegning
   for (int i = 0; i < 6; i++){
       cout << *lpek << endl;lpek++;
   }
}
```

Vad pekar **lpek** på nu då?

Strängar och pekare

Strängfunktioner arbetar med pekare

Hur lång är en sträng?

```
#include <iostream.h>
void main(void){
   char text[81], *t = text;
  int len;
   cin.getline(text, 81); // mata in en text
   // tag reda på strängens längd
   for ( len = 0 ; *t != '\0' ; len++ )
      t++;
   cout << "Din text är " << len << " bytes lång";
}
```

Que?

Lite pyssel med loopen:

```
for ( len = 0 ; *t != '\0' ; len++ )
   t++;
```

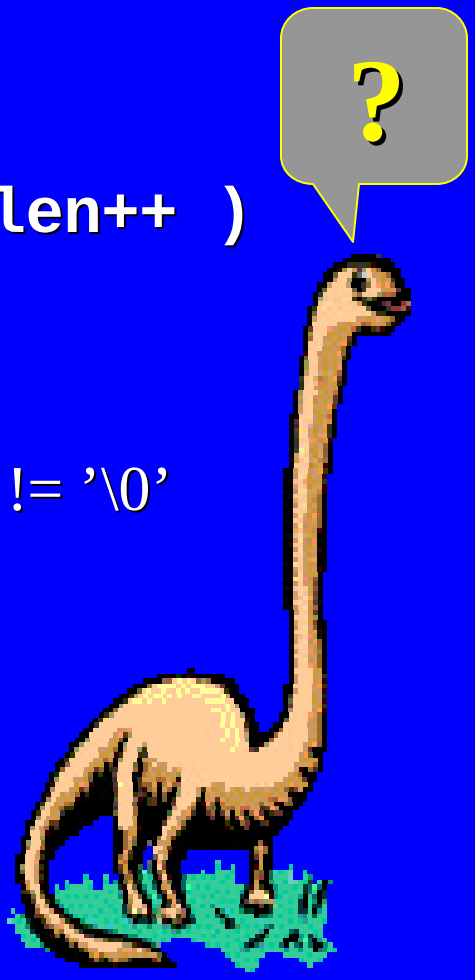
Sätt len = 0;

Loopa så länge det tecken pekaren **t** pekar på != ’\0’

I varje loop inkrementera **len**

Vad pekar t på då loopen avbryts?

Kan man reducera koden?



Mera pyssel med loopen

Loopen pågår så länge det är sant att:

```
*t != '\0' // tecknet t pekar på är inte null
```

- Villkoret kan förenklas till `*t ( null är falskt )`
- Pekaren kan inkrementeras direkt:

```
for ( len = 0 ; *t++; len++ );
```

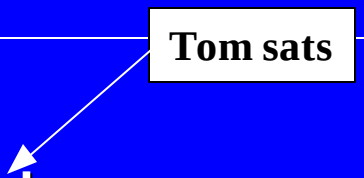
Så länge villkoret är sant utförs en tom sats.

Inkrementeringen av pekaren sker efter det logiska testet.

Alternativ

Var det smart?

```
for ( len = 0 ; *t++; len++ ) ;
```



Mjae.. Lite hackerprogrammering kan vara kul..

men minskar läsbarheten.

Loop med while är förresten kortare:

```
len = 0;
while (*t++) len++;
```

Vi återkommer till loopar och funktioner!

Varför pekare?

Om vi redan har identifierare, t.ex. :

```
char textrad[];
float volym;
```

Vad är vitsen med pekare?

Det klarnar med funktioner!

```
int minLista[400], resultat;
// fyll på listan med t.ex. inmatning
//  och skicka den till en funktion
resultat = fixaMed(minLista);
```

Man kan inte skicka med alla värden i listan till en funktion!

I stället skickar man en pekare - dvs adressen till listan.

Operatorer

För att operera på data behövs operatorer

`+ - * / osv`

Uttryck med operatorer :

x `op` *y*

`op` *x*

x `op`

`primärt uttryck`

`x` och `y` är *operander* `op är` *operator*

`tal1  381 2.5 'A' "Hej!" (i + j)` är primära uttryck

`i + j  p = q  ++k` är uttryck

Uttryck har alltid en *typ* och ett *värde*

Operatorer

Aritmetiska operatorer och uttryck

`* / + -` (jämför matematiken)

```
t1 + t2     1 - n     p * q      i/20
```

`+` och `-` finns i *unär form*:

```
+6 -1.23E3       -delvolym
```

Öknings/minskningsoperatorer (inkrementering/ dekrementering)

```
++ -- // Ökar och minskar med 1
i++; p--; // postfixoperatorer
++i; --i; // prefixoperatorer
```

Heltalsdivision

Resultatet av en heltalsdivision blir alltid ett heltal

```
int antal = 7, halva;
halva = antal/2; // halva blir 3
```

Luring:

```
int antal = 7;
float halva;
halva = antal/2; // halva är ett flyttal!
```

Det blir ändå inte 3.5!

Först görs en heltalssdivision -> HL blir 3

Därefter en typomvandlig (av 3) till **`float`**

Modulo

Operatorn **%** är modulo-operatorn som ger resten vid en heltalsdivision.

```
int tal = 17, rest;
rest = tal % 7;
cout << ”Resten blev ” << rest << endl;
```

Vad skrivs ut?

```
cout << tal % 1 << ’\t’ << tal % 17 << endl;
```

Vad skrivs ut?

```
cout << ”8 modulo 8 = ” << 8 % 8 << endl;
```

Vad skrivs ut?

The End :)