Pekare

Pointers

Dagens kluringar

```
int a=5;
int f(int b)
{
    a--;
    b++;
    return b;
}
int main()
{
    int a=3;
    printf("%d,",f(a));
    printf("%d",a);
    return 0;
}
```

```
int f(int a)
{
    if(a==1)
        return a;
    else
        return a+f(a-1);
}

int main()
{
    printf("%d",f(4));
    return 0;
}
```

Pekare

- Vad är det och hur deklarerar vi en
- Adressoperatorn och värdeoperatorn
- Användning
- Array som parameter

Vad är en pekare?

- Minnet i en dator är uppdelat i bytes. En byte består av 8 bitar. En bit kan vara antingen 1 eller 0.
- Varje byte har en unik minnes-adress
- En variabel sparas i en eller flera efterföljande byte
- En pekarvariabel är en variabel som kan innehålla minnesadressen till en variabel, vi säger att den pekar på variabeln

Adress	Innehåll
0	00000000
1	01001011
2	00000000
3	00000101
4	01001111

Ett exempel

```
int a=5;
int *p;  //p deklareras som en pekare, dvs en variabel som kan
         //innehålla adressen till en variabel (här int-variabel)
p=&a;//vi lagrar a's adress i p
```

- Vid en körning av programmet reserveras byte med adress 2 och 3 för variabeln a. Pekaren p får nu värdet 2

Adress	Innehåll
0	00000000
1	01001011
2	00000000
3	00000101
4	01001111

Deklaration

- En pekare kan endast peka på en viss sorts variabel och deklareras enligt:

```
int *p;
float *q;
char *r,*s;
```

- Och kan då endast peka på (innehålla adressen till) en variabel av typen int, float, char och char.
- Observera att * måste stå vid varje pekare när flera deklareras på samma rad

Adressoperatorn

- Med hjälp av adressoperatorn, &, kan vi komma åt en variabels minnesadress:

```
int a=5;
int *p,*q;
p=&a;//p pekar på a
q=p;//båda innehåller nu samma minnesadress (pekar på
      //a)
```

Värdeoperatorn (dereference or indirection operator)

- När vi väl har en pekare vill vi kunna komma åt variabeln den pekar på och det gör vi med värdeoperatorn *

```
int a=5;
int *p,*q;
p=&a;
q=p;
printf("%d",*p);//a's värde, dvs 5 skrivs ut
*p=8;//a's värde har nu ändrats till 8
printf("%d,%d,%d",*p,*q,a);//a's värde, dvs 8 skrivs ut 3ggr
```

- Vad betyder *&a?

Tilldelning

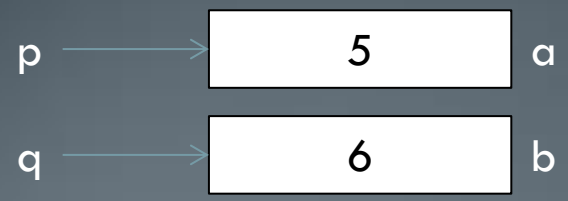
int a=5,b=6;
int *p=&a,*q=&b;
Givet koden vad händer om vi gör:

q=p; eller om vi istället gör *q=*p;

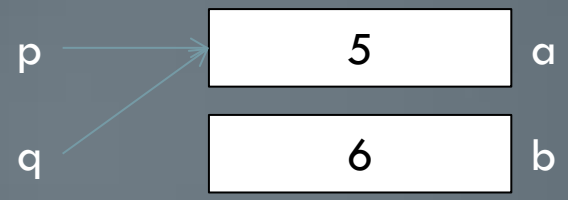
Vad blir resultatet i de två fallen om fortsättningen är:

*p=8;

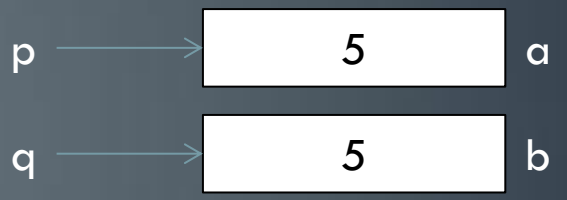
printf(”%d”,*q);



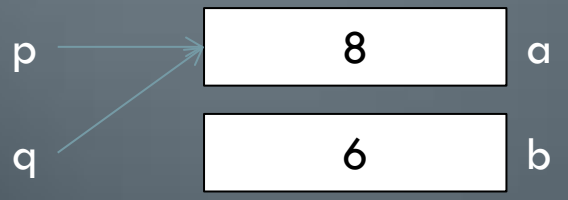
q=p;



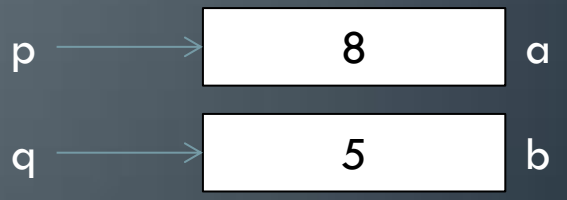
*q=*p;



*p=8;



*p=8;



Användning av pekare

- Då vet ni vad pekare är och att det är lite krångligt att använda dem men vad i hela…ska man ha dem till.
- Faktum är att pekare är användbara på en mängd olika sätt. Arrays i C är implementerade med pekare och man kan ibland vinna på att hantera dem som pekare. Vi kommer inte att använda oss direkt av detta i denna kurs men det kommer att märkas då och då.
- Pekare är också nödvändiga om vi vill kunna allokera minne dynamiskt efter behov medans ett program kör. Inte heller detta hinner vi med i denna kurs.
- Vi ska använda pekare för ett syfte endast i denna kurs: för att kunna ändra en variabels värde i en funktion.
- Däremot kommer de tillbaka i senare kurser så det är viktigt att ni får en första förståelse.

Variabler kopieras i funktionsanrop! (passed by value)

- När vi skickar med en variabel till en funktion skickar vi värdet på variabel, dvs variabeln kopieras till minnesplatsen för funktionens parameter på stacken.
- Ändrar vi parameterns värde i funktionen påverkas inte variabeln i huvudprogrammet

```
void dubbla(int tal){
     tal=tal*2;}
int main(){
     int a=5;
     dubbla(a);
     printf("%d",a);//skriver ut 5
     return 0;}
```

- Oftast är det helt ok eller till och med bra men hur gör vi om vi vill ändra variabeln i huvudprogrammet?

Pekare som parameter

- Lösningen är att vi skickar en pekare till variabeln.
- Nu kan vi komma åt variabelns värde. Pekaren som vi skickar kopieras men en kopierad minnesadress pekar ju fortfarande på samma variabel!

```
void dubbla(int *tal){
      *tal=(*tal)*2;}
int main(){
      int a=5;
      dubbla(&a);
      printf("%d",a);//skriver ut 10
      return 0;}
```

- Parentesen runt *tal behövs egentligen inte men det blir tydligare och är man osäker på prioritetsordningen är det alltid bättre att vara på den säkra sidan.
- Det är ju därför vi har skrivit: scanf(”%d”,&tal)
  scanf behöver en pekare till tal för att kunna ändra värdet på tal

Array som parameter

- När vi skickar en array som parameter skickas faktiskt egentligen en pekare till första elementet. Detta gör att vi kan ändra i en array i en funktion utan att göra något speciellt.

```
void dubbla(int tal[], int antal)
{
    int i;
    for(i=0;i<antal;i++)
        tal[i]=tal[i]*2;
}

int main()
{
    int v[]={1,2,3,4};
    dubbla(v,4);
    int i;
    for(i=0;i<4;i++)
        printf("%d,",v[i]);
    return 0;
}
```

Hålla tungan rätt i mun

- Som ni har sätt är principen för pekare enkel men det gäller att vara ordentlig och tänka rätt. Vad händer t.ex om man vill att en funktion ska ändra vad en pekare pekar på? Jo det stämmer, vi måste skicka en pekare till en pekare. Förhoppningen är att ni inte ska behöva detta i denna kurs då det är något som kan få även erfarna programmerare lite snurriga.
- Däremot kommer ni när ni jobbar med lab 3 att behöva skicka en pekare till en funktion och i denna funktion behöva skicka pekaren vidare. Låt oss därför titta på ett exempel

Ett exempel

```
void laggtill(int *antal)
{
    (*antal)++;
}

void meny(int *antal)
{
    laggtill(??????);
}

int main()
{
    int antal=5;
    meny(&antal);
    printf("%d",antal);
    return 0;
}
```

Studieanvisningar F7

- Gör K11 E1, E2, E3, E4, E6. Skriv också huvudprogram som testar att dina funktioner fungerar som det är tänkt. Lek gärna lite fritt med pekare.
- Läs11.1-11.4
- Vi har medvetet skalat ner pekare till ett minimum då det är ett ganska svårt koncept när man är ny programmerare. Om du känner att du är redo är det jättebra att läsa lite mer, bland annat kapitel 12.

E-excercises, P-programming projects