Fortsättning Pekare

Ulf Assarsson

Originalslides av Viktor Kämpe

Förra föreläsningen

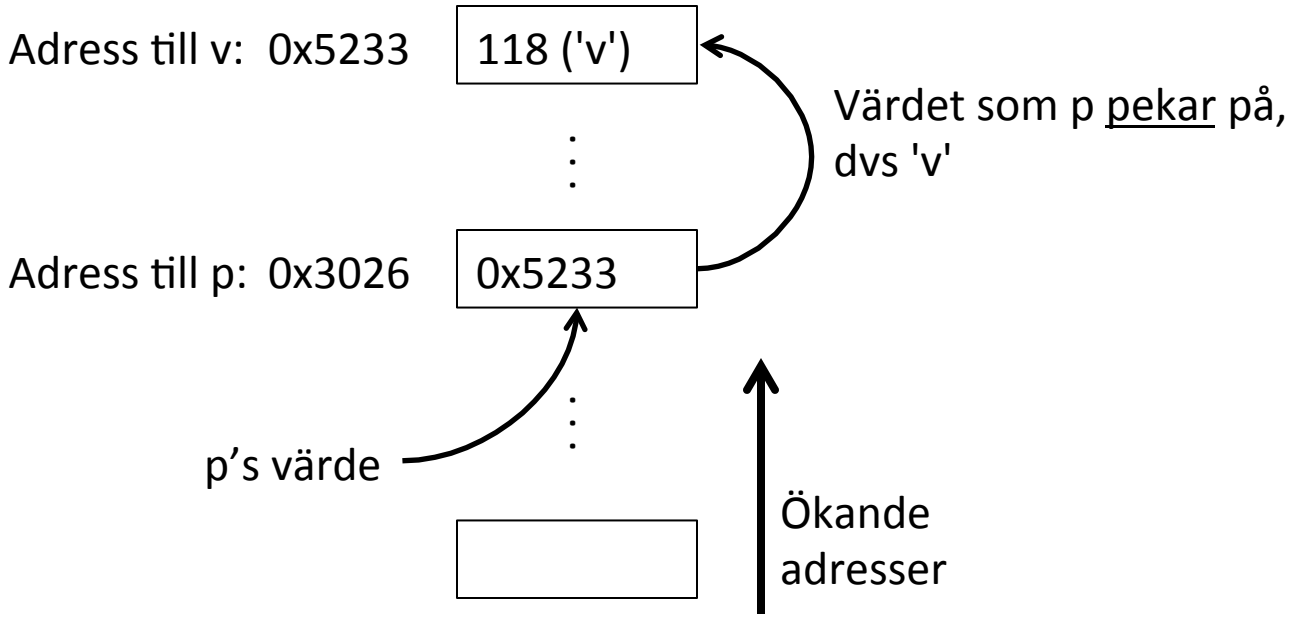
- Pekare till data
- Arrayer – fix storlek och adress
- Dynamisk minnesallokering
  - `malloc()`
  - `free()`

Pekare - sammanfattning

```
&a -> Adress till variabel a. Dvs minnesadress som a är lagrat i.
a  -> variabelns värde (t ex int, float eller en adress om a är pekarvaribel)
*a -> Vad variabel a pekar på (här måste a's värde vara en giltig
      adress och a måste vara av typen pekare)
```

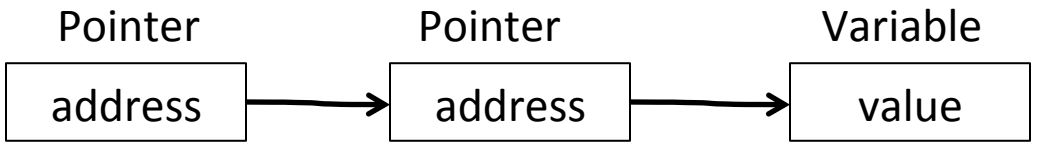
Exempel för pekarvariabel p:

```
char  c = 'v';
…
char* p = &c;
```



Pekare till pekare

```
char *p1, *p2, *p3;
char **pp;

pp = &p1;
```




```
// Annat exempel. Funktion som allokerar minne dynamiskt, t ex för elaka fiender i ett spel
void allocateEnemyArray(struct Enemy **pp, int n)
{
  *pp = (struct Enemy *)malloc(n * sizeof(struct Enemy));
}

int main()
{
  struct Enemy *pEnemies = NULL;
  allocate(&pEnemies, 100);
  // Game logic
  free(pEnemies);
}
```

Kul med pekare – vad skrivs ut?

```
#include <stdio.h>
#include <conio.h>
char * s1 = "Emilia"; // variabeln s1 är en variabel som går att ändra, och vid
                      // start tilldelas värdet av adressen till 'E'.
char s2[] = "Roger";  // värdet på s2 känt vid compile time. s2 är konstant, dvs
                      // ingen variabel som går att ändra. Är adressen till 'E'.
int main()
{
          printf("Kul med pekare\n");

          char **pp, *p = s1;
          pp = &p;
          printf("p = %s\n", p);
          printf("*pp = %s\n", *pp);

          *pp = s2;         // ändrar p till s2
          printf("p = %s\n", p);

          **pp = 'J';       // ändrar s2[0] till 'J'
          *(*pp+2) = 'k';   // ändrar s2[2] till 'k'
          printf("%s\n", p);

          (*pp)[0] = 'T';   // ändrar s2[0] till 'T'
          printf("%s\n", p);

          getch();
          return 0;
}
```

Array av arrayer

```c
#include <stdio.h>

char *fleraNamn[] = {"Emil", "Emilia", "Droopy"};

int main()
{
    printf("%s, %s, %s\n", fleraNamn[2], fleraNamn[1], fleraNamn[0]);

    return 0;
}
```

```
Droopy, Emilia, Emil
```

Array av arrayer

```
#include <stdio.h>

int arrayOfArrays[3][4] = { {1,2,3,4}, {5,6,7,8}, {9,10,11,12} };

int main()
{
    int i,j;
    for( i=0; i<3; i++) {
        printf("arrayOfArray[%i] = ", i);
        for ( j=0; j<4; j++)
            printf("%i ", arrayOfArrays[i][j]);
        printf("\n");
    }

    return 0;
}
```

Pekare till portar och funktioner

Denna föreläsning handlar om pekare till

- Portar
- Funktioner
- Sammansatta datatyper (`struct`)

Absolutadressering

- Vid portadressering så kan vi ha en absolut adress (t ex 0x400).

Absolutadressering

```
0x400                          // ett hexadecimalt tal
(unsigned char*)0x400          // en unsigned char pekare som pekar på adress 0x400
*((unsigned char*)0x400)       // dereferens av pekaren

// läser från 0x400
value = *((unsigned char*)0x400);

// skriver till 0x600
*((unsigned char*)0x600) = value;
```

Läsbarhet med typedef

```
0x400
(unsigned char*)0x400
*((unsigned char*)0x400)

// läser från 0x400
value = *((unsigned char*)0x400);
```

```
typedef unsigned char* port8ptr;
#define INPORT_ADDR 0x400
#define INPORT *((port8ptr)INPORT_ADDR)

INPORT_ADDR
(port8ptr)INPORT_ADDR
INPORT

// läser från 0x400
value = INPORT;
```

`typedef` förenklar/förkortar uttryck, vilket kan öka läsbarheten.

```
typedef unsigned char* port8ptr;
```

typ alias/typnamn

Volatile qualifier

```
char * inport = (char*)0x400;

void foo(){

    while(*inport != 0)
    {
        // ...
    }
}
```

En kompilator som optimerar kanske bara läser en gång (eller inte alls om vi aldrig skriver till adressen från programmet).

Volatile qualifier

```
volatile char * inport = (char*)0x400;

void foo(){

    while(*inport != 0)
    {
        // ...
    }
}
```

`volatile` hindrar vissa optimeringar, då kompilatorn måste anta att innehållet på adressen kan ändras utifrån.

[portadressering i XCC12]

Funktionspekare

```c
#include <stdio.h>

int square(int x)
{
    return x*x;
}

int main()
{
    int (*fp)(int);

    fp = square;

    printf("fp(5)=%i \n", fp(5));
    return 0;
}
```

En funktionspekare

```
fp(5)=25
```

Funktionspekare

```
int (*fp)(int);
```

Funktionspekarens typ bestäms av:

- Returtyp.
- Antal argument och deras typer.

Funktionspekarens värde är en adress.

Likheter assembler – C

```
utport      EQU         $400

            ORG         $1000
start:
            LDAA        #1

loop_start:
            STAA        utport
            JSR         delay

            LSLA
            BEQ         start
            BRA         loop_start


delay:
            LDAB        #$FF
loop_delay:
            DECB
            BNE         loop_delay
            RTS

var1        RMB         2
```

Både funktioner och globala variabler har adresser i minnet, men vi använder symboler.

```
int var1;


void delay()
{
    unsigned char tmp = 0xFF;
    do {
        tmp--;
    } while(tmp);
}
```

[exempel på funktionspekare]

Sammansatta datatyper

En så kallad `struct` (från eng. *structure*)

- Har en/flera medlemmar.
- Medlemmarna kan vara av
  - bas-typ (t ex `int`, `float`)
  - egendefinerad typ (t ex en annan `struct`).
  - Pekare (även till funktioner och samma `struct`)

Användning av `struct`

```
#include <stdio.h>

char* kursnamn = “Programmering av inbyggda system";

struct Course {
    char* name;
    float credits;
    int numberOfParticipants;
};

int main()
{
    struct Course pis;

    pis.name = kursnamn;
    pis.credits = 7.5f;
    pis.numberOfParticipants = 110;

    return 0;
}
```

Definition av strukturen

Deklaration av variabeln `pis`

Access till medlemmar via .-operatorn

Initieringslista

```
struct Course {
    char* name;
    float credits;
    int numberOfParticipants;
};

struct Course kurs1 = {"PIS", 7.5f, 110};
struct Course kurs2 = {"PIS", 7.5f};
```

En `struct` **kan** initieras med en initieringslista. Initieringen sker i samma ordning som deklarationerna, men alla medlemmar måste inte initieras.

Pekare till struct

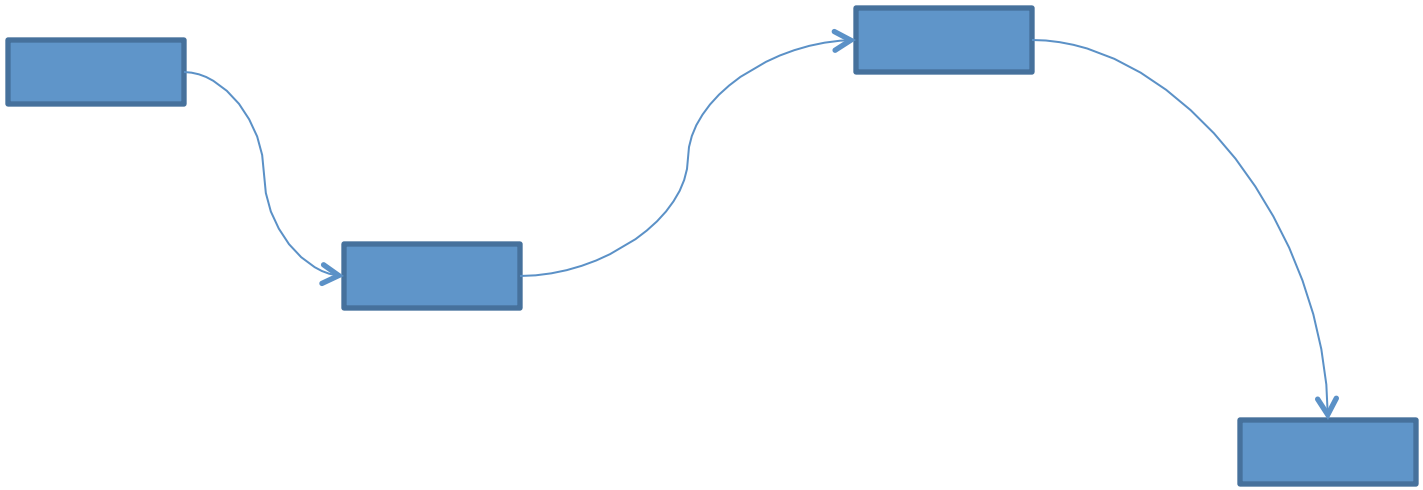
```c
#include <stdio.h>

char* kursnamn = "Programmering av inbyggda system";

struct Course {
    char* name;
    float credits;
    int numberOfParticipants;
};

int main()
{
    struct Course *ppis;
    ppis = (struct Course*)malloc(sizeof(struct Course));

    (*ppis).name = kursnamn;
    ppis->name   = kursnamn;
    ppis->credits = 7.5f;
    ppis->numberOfParticipants = 110;

    free(ppis);
    return 0;
}
```

Access till medlemmar via -> operatorn

[struct-exempel]

Länkade datastrukturer

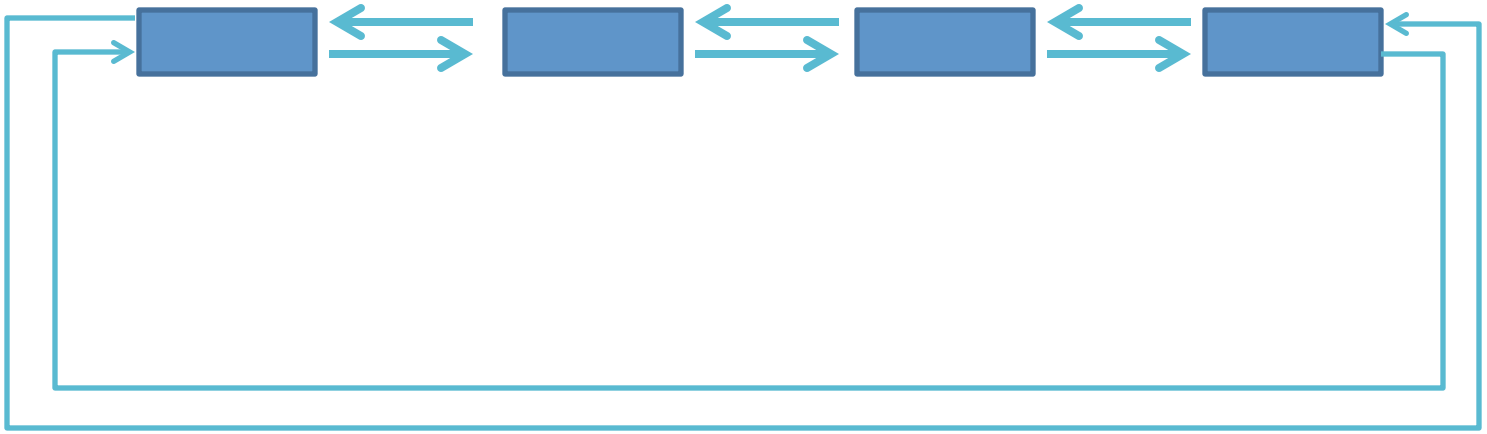
- Ex. Länkad lista:



Elementen ligger inte på konsekutiva adresser, utan listans minnesstruktur bestäms av pekare.

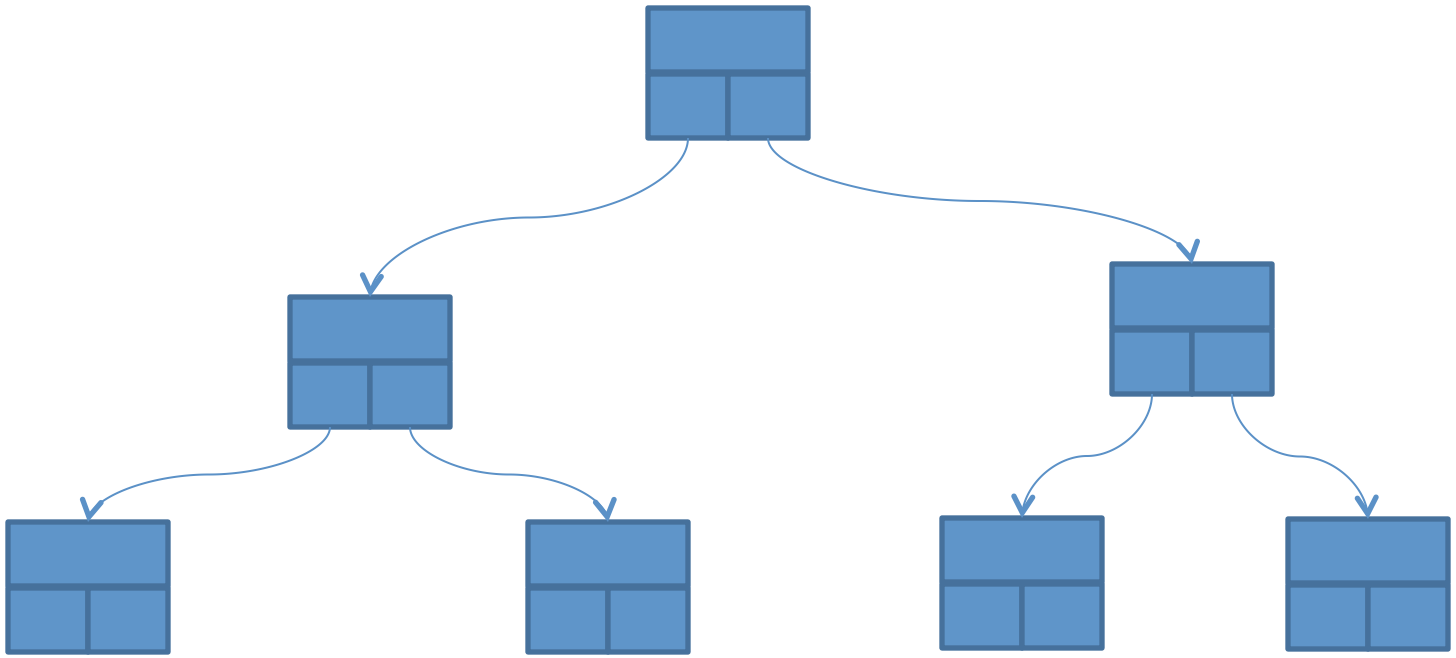
Länkade datastrukturer

- Ex. Cyklisk dubbellänkad lista:



Länkade datastrukturer

- Ex. träd:



Laboration 3 – enkellänkad lista

- En lista av element som är
  - Dynamiskt allokerade.
  - En `struct`.
- Varje element innehåller
  - En pekare till nästa element.
  - En prioritet.
  - En pekare till data (en sträng i detta fall).