Fortsättning Pekare

Viktor Kämpe

Förra föreläsningen

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

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 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 och 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};
```

initieringslista

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

Pekare till `struct`

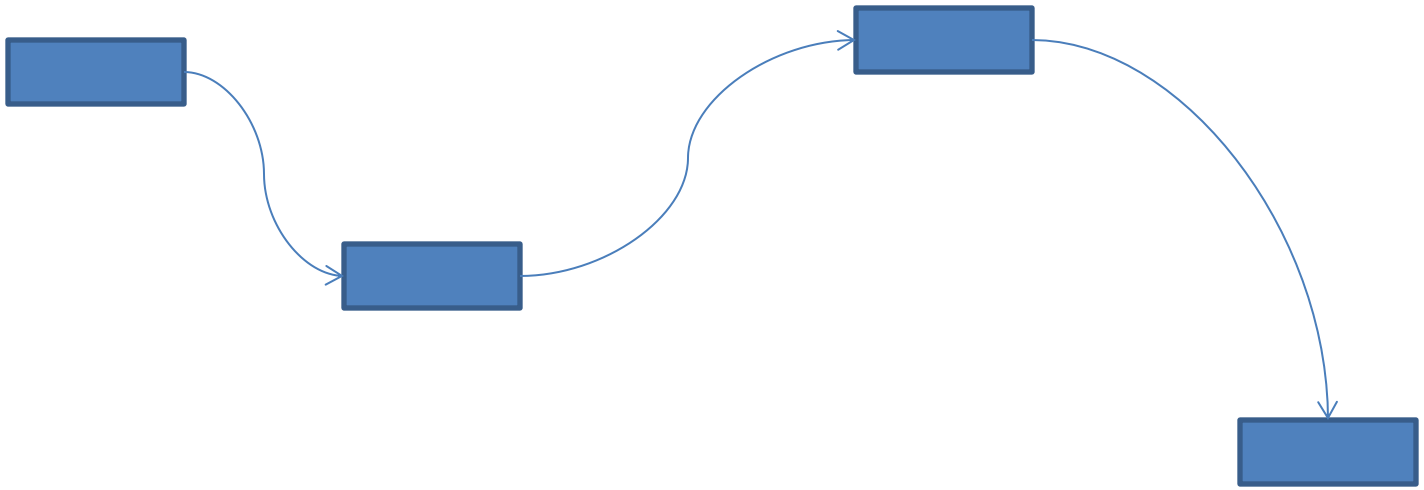
```c
#include <stdio.h>

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

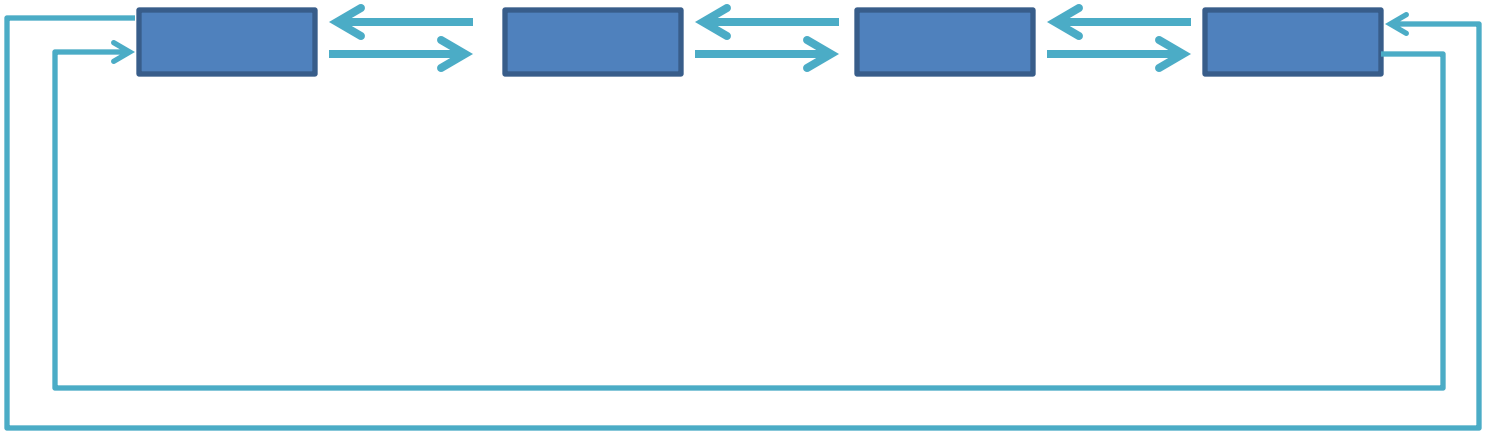
- Ex. Länkad lista:



Elementen ligger inte på konsekutive adress, utan dess struktur bestäms av pekare.

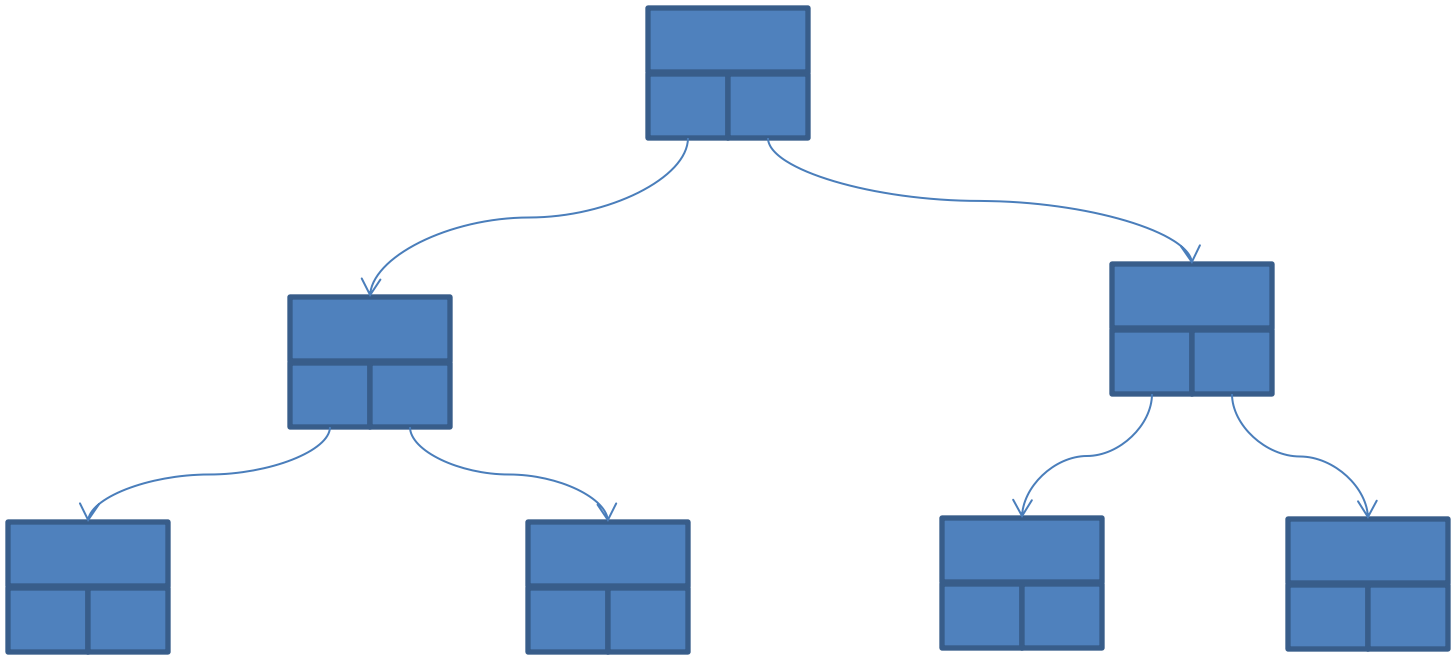
Länkade datastrukturer

- 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).