CHALMERS
UNIVERSITY OF TECHNOLOGY

Maskinorienterad Programmering

C-Programming part 2

Erik Sintorn
erik.sintorn@chalmers.se

Original slides by Pedro Trancoso and Ulf Assarsson

Innehåll

- Pekare
- Absolut addresering (portar)
- typedef, volatile, #define
- Arrays av pekare, arrays av arrays

Förra C lektionen

- C-syntax
- Visibility, Typkonvertering, Arrays, Strängar
- Programstruktur, kompilering och länkning
- Bitwise operationer

Varför behöver vi pekare?

```
int TryToChange(int value, int access)
{
    if(state == READ_ONLY) { return ERROR; }
    else {
        value = 5;
        return SUCCESS;
    }
}

int main()
{
    int value = 10;
    int access = READ_WRITE;
    TryToChange(value, access);
    printf("Value: %i\n", value);
}
```

```
Output: Value: 10
```

Funktionen tar emot en *pekare*:
minnesadressen till en int

```
int TryToChange(int * value, int access)
{
    if(state == READ_ONLY) { return ERROR; }
    else {
        *value = 5;
        return SUCCESS;
    }
}

int main()
{
    int value = 10;
    int access = READ_WRITE;
    TryToChange(&value, access);
    printf("Value: %i\n", value);
}
```

Dereferencing: Ändra värdet som "value" *pekar på*

Vi skickar med *adressen* till "value"

```
Output: Value: 5
```

Varför behöver vi pekare?

```
int IsEvil(Monster m)
{
    return m.is_evil;
}

int main()
{
    Monster monsters[1000];
    for(int i=0; i<1000; i++) {
        if(IsEvil(monster[i]))
            printf("Monster %i is evil!\n",
    }
}
```

```
int IsEvil(Monster *m)
{
    return (*m).is_evil;
}

int main()
{
    Monster monster[1000];
    for(int i=0; i<1000; i++) {
        if(IsEvil(&monster[i]))
            printf("Monster %i is evil!\n", i);
    }
}
```

Funktionen tar emot en *pekare*: *minnesadressen* till ett monster

Dereferencing: Använd objektet som "monster" *pekar på*

Vi skickar med *adressen* till ett monster

Pekare

- `int monkey;` **// Det finns en oinitialiserad variabel, "monkey", i minnet som innehåller en integer.**
  `monkey = 123;` **// Sätt den integern till 123**

- `int * elephant;` **// Det finns en onitialiserad variable, "elephant", i minnet som innehåller**
  // en *pekare* (minnesadress). Adressen pekar på en integer.
  `elephant = 123;` **// Sätt *addressen* till 123.**
  `*elephant = 456;` **// Tyd de fyra bytesen på address 123 som en integer, och sätt dem till 456.**

Pekare och indexering / aritmetik

```
int main()
{
    int an_array[] = { 0,

    int * a_pointer = an_a
    printf("%i\n", a_point

    a_pointer = &an_array[0];
    printf("%i\n", a_pointer[4

    a_pointer = &an_array[1];
    printf("%i\n", a_pointer[4
}
```

```
int * pointer0 = (int *) 0x1000;
int * pointer1 = pointer0 + 1;
int * pointer2 = &pointer0[1];
printf("0x%p, 0x%p, 0x%p\n", pointer0, pointer1, pointer2);
// Output: 0x0000000000001000, 0x0000000000001004, 0x0000000000001004
```

```
// Get a pointer to first element of array
short int * pointer0 = (short int *) 0x1000;
short int * pointer1 = pointer0 + 1;
short int * pointer2 = &pointer0[1];
printf("0x%p, 0x%p, 0x%p\n", pointer0, pointer1, pointer2);
```

Vad är skillnaden på en pekare och en array?

```c
#include <stdio.h>

int main()
{
    int an_array[10];
    int * a_pointer = an_array;
    for(int i=0; i<10; i++)
    {
        if(an_array[i] != a_pointer[i])
            printf("This won't happen.\n"
        }
    }
}
```

```c
void f(int * ptr)
{
}

int main()
{
```

```c
int main()
{
    int an_array[10];
    int another_array[10];
    int * a_pointer = an_array; // This is fine
    another_array = an_array;   // This does not compile.
}
```

Sammanfattning pekarsyntax

`t *p;`	p declared of type “pointer to type `t`“
`p = 0;`	p becomes a null pointer (pointer to nothing!)
`p = &v;`	p is assigned the address of variable v
`*p`	means “content of what p points to”
`p1 = p2;`	p1 will point to the same pointed by p2
`*p1 = *p2;`	content of what is pointed by p1 becomes the same as the content of what p2 points to.

Mentimeterspelet!

```c
#include <stdio.h>

int main()
{
    int a[] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
    int * b = &a[3];
    char * c = b;
    c += 4;
    int d = *c;
    // What's the output?
    printf("0x%x\n", d);

    // Överkurs
    c[1] = 0x5;
    int e = *((int *)c);
    printf("0x%x\n", e);
}
```

typedef

```c
int main()
{
    unsigned short int * ptr0, ptr1, ptr2, ptr3;
    int a = *ptr0;
    int b = *ptr1; // ERROR!!!
}
```

```c
int main()
{
    unsigned short int * ptr0, * ptr1, * ptr2, * ptr3;
    int a = *ptr0;
    int b = *ptr1; // OK
}
```

```c
typedef unsigned short int uint16;

int main()
{
    uint16 * ptr0, * ptr1, * ptr2, * ptr3;
    int a = *ptr0;
    int b = *ptr1; // OK
}
```

```c
typedef unsigned short int * uint16_ptr;

int main()
{
    uint16_ptr ptr0, ptr1, ptr2, ptr3;
    int a = *ptr0;
    int b = *ptr1; // OK
}
```

Absolut adressering

- Vi kan peka till en absolut adress så här:
  ```
  int * pointer = (int *)0xA0001000;
  ```

- Varför skulle vi vilja göra det?

Läsa och skriva portar

```
int main()
{
    // Write to port
    *((unsigned char *)0xA0001000) = 0xAA;
    // Read from port
    int a = *((unsigned char *)0xA0001000);
}
```

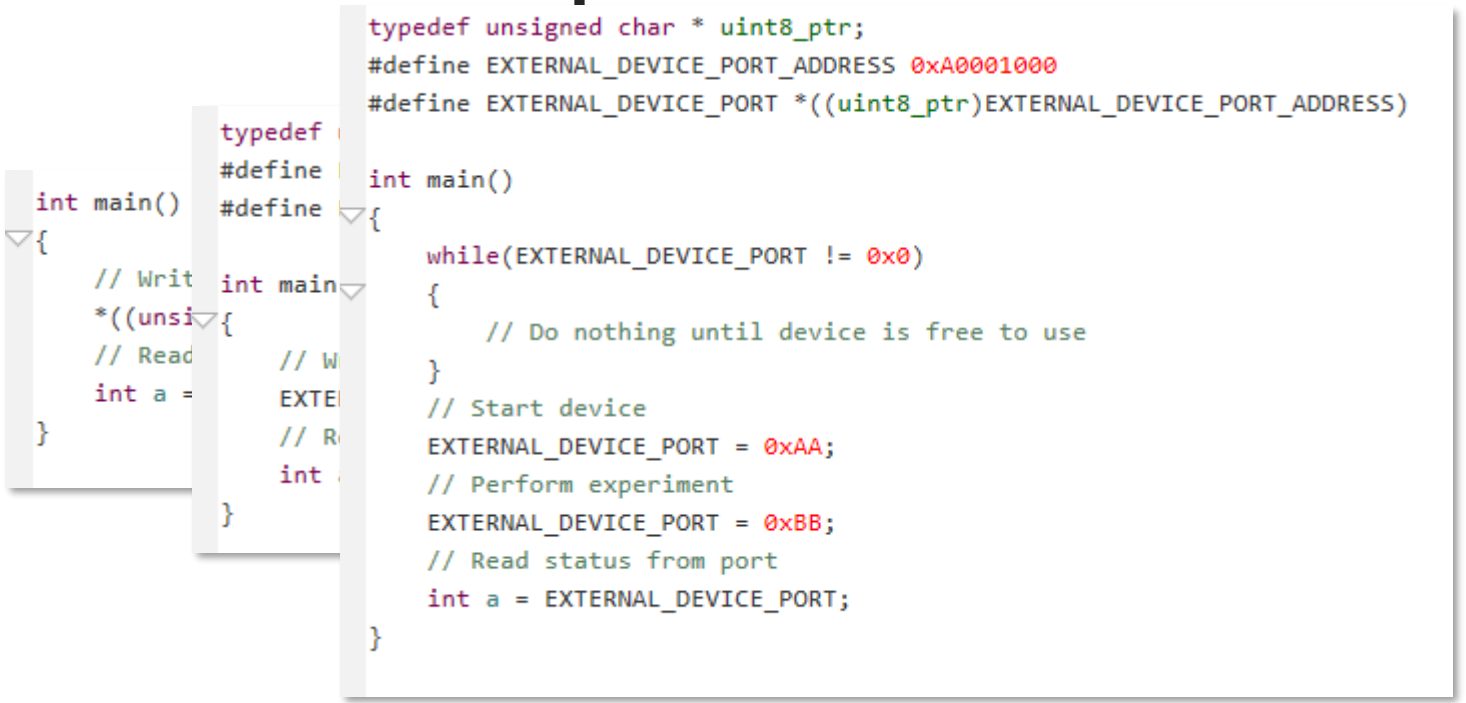
Läsa och skriva portar

```
int main()
{
    // Writ
    *((unsi
    // Read
    int a =
}
```

```
typedef unsigned char * uint8_ptr;
#define EXTERNAL_DEVICE_PORT_ADDRESS 0xA0001000
#define EXTERNAL_DEVICE_PORT *((uint8_ptr)EXTERNAL_DEVICE_PORT_ADDRESS)

int main()
{
    // Write to port
    EXTERNAL_DEVICE_PORT = 0xAA;
    // Read from port
    int a = EXTERNAL_DEVICE_PORT;
}
```

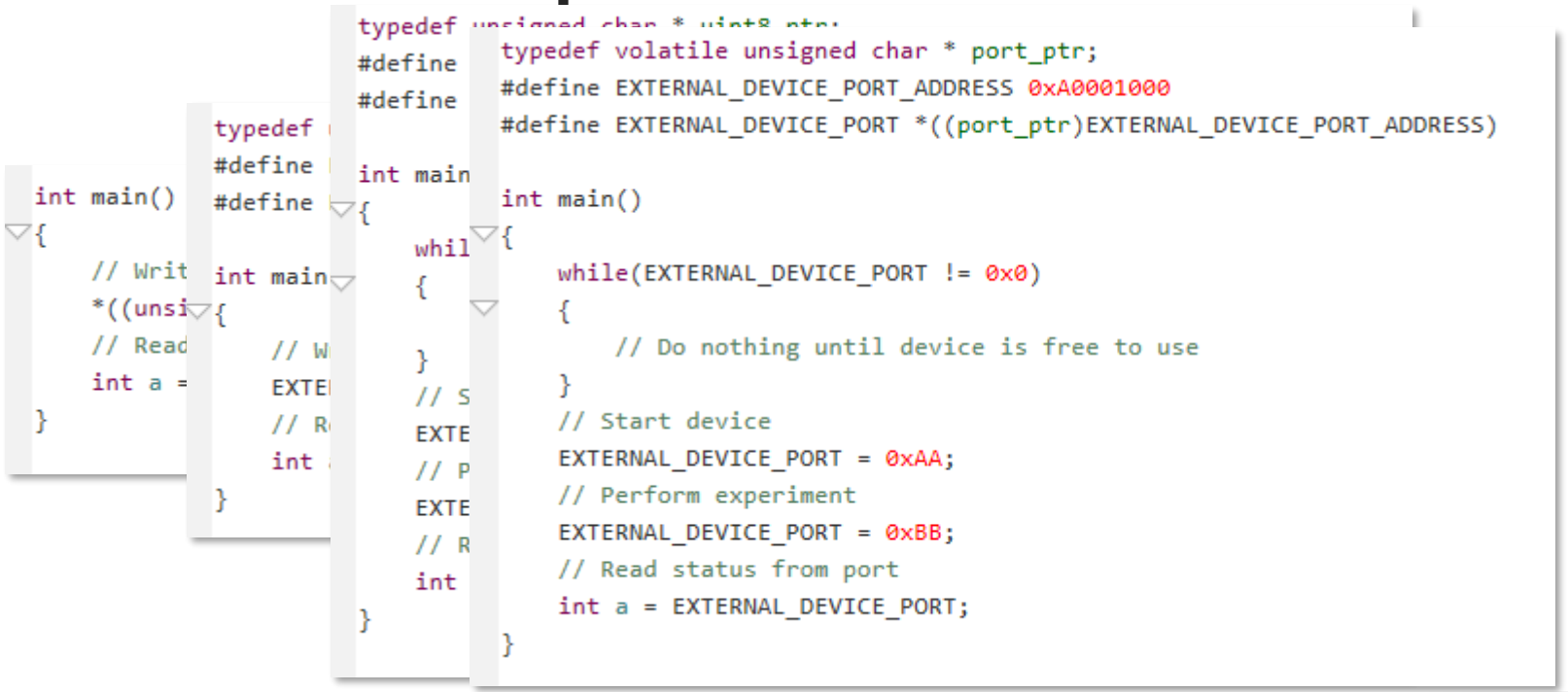
Läsa och skriva portar



```c
typedef unsigned char * uint8_ptr;
#define EXTERNAL_DEVICE_PORT_ADDRESS 0xA0001000
#define EXTERNAL_DEVICE_PORT *((uint8_ptr)EXTERNAL_DEVICE_PORT_ADDRESS)

int main()
{
    while(EXTERNAL_DEVICE_PORT != 0x0)
    {
        // Do nothing until device is free to use
    }
    // Start device
    EXTERNAL_DEVICE_PORT = 0xAA;
    // Perform experiment
    EXTERNAL_DEVICE_PORT = 0xBB;
    // Read status from port
    int a = EXTERNAL_DEVICE_PORT;
}
```

Läsa och skriva portar



```c
typedef volatile unsigned char * port_ptr;
#define EXTERNAL_DEVICE_PORT_ADDRESS 0xA0001000
#define EXTERNAL_DEVICE_PORT *((port_ptr)EXTERNAL_DEVICE_PORT_ADDRESS)

int main()
{
    while(EXTERNAL_DEVICE_PORT != 0x0)
    {
        // Do nothing until device is free to use
    }
    // Start device
    EXTERNAL_DEVICE_PORT = 0xAA;
    // Perform experiment
    EXTERNAL_DEVICE_PORT = 0xBB;
    // Read status from port
    int a = EXTERNAL_DEVICE_PORT;
}
```

Arrays of arrays

```c
int main()
{
    // Have to give size of inner
    int matrix[][3] = {{0, 1, 2}, {3, 4, 5}, {6, 7, 8}};
    printf("Middle: %i\n", matrix[1][1]);

    char short_names[][4] = {"Tor", "Ulf", "Ian"};
    printf("Middle: %s\n", short_names[1]);
}
```

```
matrix[2][1] = matrix + 2 * 3 + 1
```

Arrays of pointers

```c
int main()
{
    char *long_names[] = {
        "Alexandrovitj Bulgakowski",
        "Constantine Plumberbatch",
        "Bret",
        "Nathaniel Prescott",
        "Jacqueline Bratwurst"
    };
    printf("Middle: %s\n", long_names[2]);
}
```

```c
int main()
{
    char *long_names[] = {
        "Alexandrovitj Bulgakowski",
        "Constantine Plumberbatch",
        "Bret",
        "Nathaniel Prescott",
        "Jacqueline Bratwurst"
    };

    char * name = "Bartholomew Butterscotch";
    long_names[2] = name;

    printf("Middle: %s\n", long_names[2]);
}
```

Pointer to pointer

```c
typedef int error_code;
#define NO_ERROR 0
#define FATAL_ERROR 1

error_code GetPortAddress(int ** inport)
{
    *inport = (int *)0xA0001000;
    return NO_ERROR;
}

int main()
{
    int * inport;
    error_code status = GetPortAddress(&inport);
    if(status == FATAL_ERROR)
    {
        printf("FATAL ERROR: Evacuate!\n");
    }
}
```

Nästa lektion

- **Structs**
- **Funktionspekare**