Grundläggande C-programmering – del 4
Mer programstruktur samt Dynamisk minnesallokering

Ulf Assarsson

Läromoment:
- Synlighet – static, #extern, (inline), #if/#ifdef, #include guards,
- enum, union, little/big endian
- Dynamisk minnesallokering (malloc/free)

Kopplat till:
- Arbetsbok avsnitt: 5 grafisk display

Hemuppgifter: v4.

Föregående lektion

- Structs, array av structs, pekare till structs (pilnotation):

```
typedef struct {
    char* name;
    float credits;
} Student;

Student students[] = {{"Per", 200.0f}, {"Tor", 200.0f}, {"Ulf",  20.0f}};
Student *pMiddleStud = &students[1];
pMiddleStud->name = "Erik"; // pilnotation. Istället för (*pMiddleStud).name
```

- Portadressering med structs:

```
typedef struct {
    uint32_t    moder; ...
} GPIO;
typedef volatile GPIO* gpioptr;
#define GPIO_E (*((gpioptr) 0x40021000))
GPIO_E.moder    =  0x55555555;
```

- Funktionspekare:

```
int (*fp)(int) = add;
```

  - Funktionspekaren fp pekar på funktionen add(). Anrop: fp(5)

- structs med funktionspekare (objektorienterad stil):

```
obj[i]->draw(obj[i], …);
```

Type Union

Union allows to store different data types in the same memory location. But they will overwrite each other – so be careful. Same syntax as struct

```
union opt_name {
      int a;
      char b;
      float c;
 } x;
 x.a = 4;
 x.b = 'i';
 x.c = 3.0;
 // x.a is now = 1077936128 (=0x40400000)
```

&x.a == &x.b == &x.c
a, b och c delar samma minnesadress. D v s samma minnesadress kan adresseras på tre olika sätt via tre olika variabelnamn.

```
typedef union {
      float v[2];
      struct {float x,y;};
} Vec2f;

Vec2f pos;
pos.v[0] = 1; pos.v[1] = 2;
pos.x = 1; pos.y = 2;
```

Exempel: pos.v[0] och pos.x är samma sak. "pos.x" visar tydligt att det är x-koordinaten vi adresserar. "pos.v[i]" är dock användbart om vi vill skriva en loop över x- och y-koordinaten.
Tex:

```
Vec2f addVec(Vec2f a, Vec2f b)
{
  for(i=0; i<2; i++)
    a.v[i] += b.v[i];
  return a;
}
```

Endian

- Big endian / Little endian

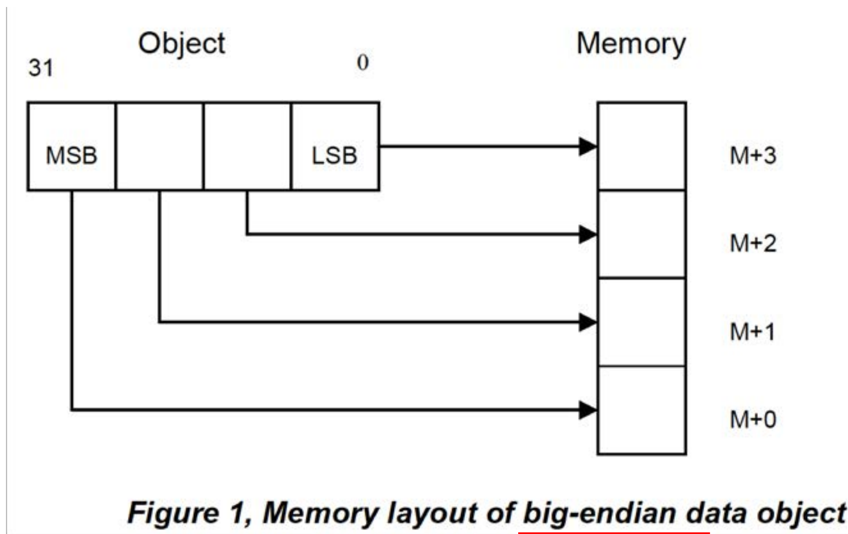

Figure 1, Memory layout of big-endian data object

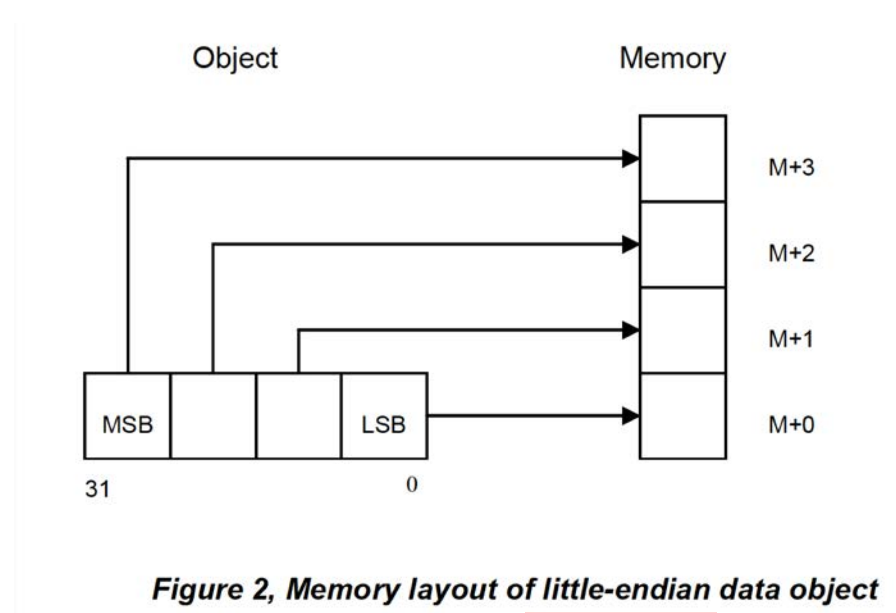

Figure 2, Memory layout of little-endian data object

Vi använder little endian. ARM Cortex-m4 klarar båda via inställning.

Byte-adressering med unions – för GPIO-porten

```
// GPIO
typedef struct _gpio {
   uint32_t   moder;
   uint32_t   otyper;
   uint32_t   ospeedr;
   uint32_t   pupdr;
   union {
      uint32_t   idr;
      struct {
         byte   idrLow;
         byte   idrHigh;
      };
   };
   union {
      uint32_t   odr;
      struct {
         byte   odrLow;
         byte   odrHigh;
      };
   };
} GPIO;
#define GPIO_D (*((volatile GPIO*) 0x40020c00))
#define GPIO_E (*((volatile GPIO*) 0x40021000))
```

GPIO Input Data Register (IDR)

offset	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	mnemonic
0x10	RESERVERADE																r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	IDR
																	0x11								0x10								

Bitar 16 tom 31 används inte och ska hållas vid sitt RESET-värde, dvs 0.

Nu kan idrHigh adresseras med:

```
byte c = GPIO_E.idrHigh;
```

Istället för med:

```
byte c = *((byte*)&(GPIO_E.idr) + 1));
```

Ett till exempel:

```
GPIO_E.odrLow &= ( ~B_SELECT & ~x);
```

Istället för:

```
*((byte*)&(GPIO_E.odr)) &=(~B_SELECT & ~x);
```

Enumerations – enum

```
enum type_name { value1, value2,..., valueN };
// type_name optional. By default, value1 = 0, value2 = value1 + 1, etc.

enum type_name { value1 = 0, value2, value3 = 0, value4 }; // helt OK.
// Ger dock med gcc värdena: 0, 1, 0, 1 – vilket kanske är ointuitivt.

enum day {monday=1, tuesday, wednesday, thursday, friday, saturday, sunday};
enum day today; // day blir en char, short eller int
today=wednesday;
printf("%d:th day",today+1); // output: "4:th day"

typedef enum { false, true } bool;
bool ok = true;
--------------------------------------------------------------------------------
#define B_E        0x40               // Uppg 29 – grafisk display - med enums
#define B_RST      0x20
#define B_CS2      0x10
#define B_CS1      8
#define B_SELECT   4
#define B_RW       2
#define B_RS       1
```

graphic_ctrl_bit_clear(B_RS | B_RW);

Kan bytas ut mot:

```
enum {B_RS=1, B_RW=2, B_SELECT=4, B_CS1=8, B_CS2=0x10, B_RST=0x20, B_E=0x40};
```

Preprocessor directives - #if, #ifdef, #ifndef

Preprocessor conditional inclusion

```
#define X  1          // syntax: #define [identifier name] [value], där [value] är optional

#if X == 0            // syntax: #if <value>, där 0=false och !0==true
    …     // any C-code
#elif X-1 == 1        // betyder else if
    …
#else
    …
#endif

#if 0     // bra för att temporärt kommentera bort stora block av kod.
 … // any C-code
#endif

#define HW_DEBUG
…
#ifdef HW_DEBUG
    … // any C-code, t ex:
    #undef SIMULATOR
#endif
```

```
void delay_500ns(void)
{
#ifndef SIMULATOR
        delay_250ns();
        delay_250ns();
#endif
}
```

Include guards

Include guards används för att automatiskt undvika att inkludera en .h-fil mer än en gång per .c-fil. Detta kan annars hända om en .c-fil inkluderar flera .h-filer som i sin tur inkluderar **samma** .h-fil.

Include guards

```
// main.c
#include "player.h"
#include "enemies.h"

void main()
{
    …
}
```

c-fil

```
// vecmath.h
#ifndef VECMATH_H
#define VECMATH_H

typedef struct {
   union {
      int v[2];
      struct {int x, y;};
   };
} Vec2i;

Vec2i add2i(Vec2i a, Vec2i b);

char isEqual(Vec2i a, Vec2i b);

#endif //VECMATH_H
```

```
// player.h
#ifndef PLAYER_H
#define PLAYER_H

#include "vecmath.h"

void movePlayer(Vec2i v);


#endif //PLAYER_H
```

```
// enemies.h
#ifndef ENEMIES_H
#define ENEMIES_H

#include "vecmath.h"

void moveEnemy(int i,
               Vec2i v);

#endif //ENEMIES_H
```

Vi bör ha include-guards på alla .h-filer – även om det i detta exemplet inte behövs för player.h samt enemies.h

Utan include guards så inkluderas vecmath.h två gånger för main.c. Andra gången kommer kompilatorn klaga på att `Vec2i` redan är definierad.

.c-filer kompileras var för sig. Preprocessorn och kompilatorn exekveras individuellt per .c-fil.

Boot Camp

[mopsimulator] C:\Users\uffe\Desktop\work\mop\mopsimulator\uppg34-graphicsdispl

File Edit View Search Workspace Build Debugger Plugins Perspective Settings PHP Help

Workspace View

uppg34-grapl Debug

- mopsimulator
  - delay-bargraph
  - startup
  - uppg27-keypad
  - uppg29-ASCIIdisplay
  - **uppg34-graphicsdisplay**
    - include
      - asciidisplay.h
      - delay.h
      - graphicsdisplay.h
      - keypad.h
      - ports.h
      - segdisplay.h
      - types.h
    - resources
    - src
      - asciidisplay.c
      - delay.c
      - graphicsdisplay.c
      - keypad.c
      - main.c
      - ports.c
      - segdisplay.c

Workspace Explorer Tabs Tabgroups CMake Help

main.c ports.h delay.h types.h

```c
//#include <stdio.h>
#include "types.h"
#include "delay.h"
#include "ports.h"
#include "graphicsdisplay.h"

/*
 *  Uppgift 34.
 *  Connect PE0-15 to graphic display. Connect Serial-communication-interface to Console
*/

static void Init( void )
{
#ifdef HW_DEBUG
   hwInit();
#endif
}

static void Exit( void )
{
 #ifdef HW_DEBUG
   exitDBG();
#endif
}

void startup(void) __attribute__((naked)) __attribute__((section(".start_section")));
void startup (void)
{
```

Boot Camp

[mopsimulator] C:\Users\uffe\Desktop\work\mop\mopsimulator\uppg34-graphicsdisp

File Edit View Search Workspace Build Debugger Plugins Perspective Settings PHP Help

Workspace View

uppg34-grapl Debug

- mopsimulator
  - delay-bargraph
  - startup
  - uppg27-keypad
  - uppg29-ASCIIdisplay
  - **uppg34-graphicsdisplay**
    - include
      - asciidisplay.h
      - delay.h
      - graphicsdisplay.h
      - keypad.h
      - ports.h
      - segdisplay.h
      - types.h
    - resources
    - src
      - asciidisplay.c
      - delay.c
      - graphicsdisplay.c
      - keypad.c
      - main.c
      - ports.c
      - segdisplay.c

Workspace Explorer Tabs Tabgroups CMake Help DbExplorer

main.c | ports.h | delay.h | types.h

```
1  #ifndef PORTS_H
2  #define PORTS_H
3
4  #include "types.h"
5
6  #define SIMULATOR
7
8  // GPIO
9  typedef struct _gpio {
10     uint32_t    moder;
11     uint32_t    otyper;
12     uint32_t    ospeedr;
13     uint32_t    pupdr;
14     union {
15         uint32_t    idr;
16         struct {
17             byte    idrLow;
18             byte    idrHigh;
19         };
20     };
21     union {
22         uint32_t    odr;
23         struct {
24             byte    odrLow;
25             byte    odrHigh;
26         };
27     };
28 } GPIO;
29 typedef volatile GPIO* gpioptr;
30 #define GPIO_D (*((gpioptr) 0x40020c00))
31 #define GPIO_E (*((gpioptr) 0x40021000))
```

Boot Camp

[mopsimulator] C:\Users\uffe\Desktop\work\mop\mopsimulator\uppg34-graphicsdispl

File Edit View Search Workspace Build Debugger Plugins Perspective Settings PHP Help

Workspace View

<global> | delay_milli(uint32 ms)

main.c | ports.h | delay.h | types.h

uppg34-grapl | Debug

- mopsimulator
  - delay-bargraph
  - startup
  - uppg27-keypad
  - uppg29-ASCIIdisplay
  - **uppg34-graphicsdisplay**
    - include
      - asciidisplay.h
      - delay.h
      - graphicsdisplay.h
      - keypad.h
      - ports.h
      - segdisplay.h
      - types.h
    - resources
    - src
      - asciidisplay.c
      - delay.c
      - graphicsdisplay.c
      - keypad.c
      - main.c
      - ports.c
      - segdisplay.c

Workspace | Explorer | Tabs | Tabgroups | CMake Help | DbExplorer

```
1  #ifndef DELAY_H
2  #define DELAY_H
3
4  #include "types.h"
5
6  void delay_250ns(void);
7  void delay_500ns(void);
8  void delay_micro(uint32 us);
9  void delay_milli(uint32 ms);
10
11
12 #endif // DELAY_H
```

[mopsimulator] C:\Users\uffe\Desktop\work\mop\mopsimulator\uppg34-graphicsdis

File Edit View Search Workspace Build Debugger Plugins Perspective Settings PHP Help

Workspace View

uppg34-grapl | Debug

- mopsimulator
  - delay-bargraph
  - startup
  - uppg27-keypad
  - uppg29-ASCIIdisplay
  - **uppg34-graphicsdisplay**
    - include
      - asciidisplay.h
      - delay.h
      - graphicsdisplay.h
      - keypad.h
      - ports.h
      - segdisplay.h
      - types.h
    - resources
    - src
      - asciidisplay.c
      - delay.c
      - graphicsdisplay.c
      - keypad.c
      - main.c
      - ports.c
      - segdisplay.c

main.c | ports.h | delay.h | types.h

```c
8   #ifndef TYPES_H
9   #define TYPES_H
10
11  typedef unsigned char           byte;
12  typedef unsigned short          word;
13  typedef unsigned int            dword;
14  typedef int                     bool;
15  typedef signed char             int8_t;
16  typedef unsigned char           uint8_t;
17  typedef signed short int        int16_t;
18  typedef unsigned short int      uint16_t;
19  typedef signed int              int32_t;
20  typedef unsigned int            uint32_t;
21  typedef unsigned long long      uint64_t;
22  typedef long long               int64_t;
23  typedef int8_t                  int8;
24  typedef uint8_t                 uint8;
25  typedef int16_t                 int16;
26  typedef uint16_t                uint16;
27  typedef int32_t                 int32;
28  typedef uint32_t                uint32;
29  typedef int64_t                 int64;
30  typedef uint64_t                uint64;
31  typedef unsigned char           uchar_t;
32  typedef uint32_t                wchar_t;
33  typedef uint32_t                size_t;
34  typedef uint32_t                addr_t;
35  typedef int32_t                 pid_t;
36
37  #endif
```

Extern

extern i C säger att symbolen kommer från en annan fil.

```
// main.c
#include <stdio.h>

extern int g_var;
void myFunc();

void main()
{
    g_var = 5;
    myFunc();
}
```

Ren deklaration – utan definition.
g_var definerad senare – t ex i annan .c-fil
extern ej nödvändig för funktioner, prototypen säger: finns vid länkning.

```
// myFunc.c
int g_var;
void myFunc()
{
    printf("hej");
};
```

Extern

Poäng: Om ni kodar ett större program för lab5 så vill ni nog inte ha alla globala objekt/variabler deklarerade i main-filen. Då måste ni använda extern.

extern i C säger att symbolen kommer från en annan fil.

```
// main.c
#include "fractalmountain.h"
…

POBJECT objects[] = {&player, &fmountain};
unsigned int nObjects = 2;

void main()
{
    …
}
```

```
// fractalmountain.h
#ifndef FRACTALMOUNTAIN_H
#define FRACTALMOUNTAIN_H

#include "object.h"
#include "types.h"

extern OBJECT fmountain;

#endif //FRACTALMOUNTAIN_H
```

```
// fractalmountain.c
#include "fractalmountain.h"

OBJECT fmountain = {
  0,               // geometri - ingen
  0,0,             // riktningsvektor
  0,0,             // initial startposition
  drawMountain,    // draw method
  0,               // clear_object – unused
  moveMountain,    // move method
  set_object_speed // set-speed method
};
```

extern = declare without defining

Utan extern skulle vi skapa en **ny** variabel.

C – synlighet för deklarationer

Alla deklarationer (variabler, funktioner) och även uttryck som typedefs + defines är synliga först nedanför dem – ej ovanför dem.

Källkodsfiler processas uppifrån och ned. Exempel:

```
void fkn1(short param)
{
    …
    if(…) {
       return fkn2('a'); // fkn2 är här ännu okänd för C-kompilatorn så detta ger
    }                    // kompileringsfel.
    …
    return;
}

void fkn2(char c)
{
    …
    return;
}
```

C – synlighet för deklarationer

Alla deklarationer (variabler, funktioner) och även uttryck som typedefs + defines är synliga först nedanför dem – ej ovanför dem.

Källkodsfiler processas uppifrån och ned. Exempel:

```
void fkn2(char c);       // Fix för att göra deklarationen av fkn2 känd redan här.
void fkn2(char c);       // Vi får ha med deklarationen hur många ggr vi vill
void fkn1(short param)
{
    …
    if(…) {

       return fkn2('a'); // Här är nu fkn2 känd
    }
    …
    return;
}

void fkn2(char c)       // Definitionen av fkn2
{
    …
    return;
}
```

C – synlighet för deklarationer

Alla deklarationer (variabler, funktioner) och även uttryck som typedefs + defines är synliga först nedanför dem – ej ovanför dem.

Källkodsfiler processas uppifrån och ned. Exempel:

```
void fkn2(char c);        // Fix för att göra deklarationen av fkn2 känd redan här.
                          // Vi får ha med deklarationen hur många ggr vi vill
void fkn1(short param)
{
    void fkn2('a');       // Vi kan även lägga den här istället
    if(…) {
       void fkn2('a');    // eller här
       return fkn2('a'); // Här är nu fkn2 känd
    }
    …
    return;
}

void fkn2(char c)        // Definitionen av fkn2
{
    …
    return;
}
```

Synlighet av symboler

- Alla symboler synliga per-default i C för länkaren.

static

Statisk variabel – ligger i programmets datasegment på fix address under hela programmets livstid.
Dynamisk variabel – ligger på stacken, dvs flyktigt endast under funktionsanropet.

`static` i C kan göra två saker:

1. Begränsar synligheten av **global** variabel eller funktion till endast dess .c fil. Unviker därmed namnkonflikter mellan olika .c filer vid länkningen:
   - `static int tmp;`
   - ```
     static void f()
     {
         …
     }
     ```

2. Allokerar minne för **lokal** variabel **statiskt**, dvs i programmets datasegment istället för dynamiskt/flyktigt på stacken. Gör att variabeln behåller sitt värde mellan funktionsanrop. Variabeln har fortfarande endast synlighet inom dess scope (funktion eller närmast omgivande {…}-block).
   - ```
     void f()
     {
         static int tmp;
         …
     }
     ```

Se även Hemuppgifter-förel. 4

Static variant 1

```
static int var;
```

Variabeln *var* kan ej ses från andra filer, oavsett om man använder `extern`.

Static variant 2

```
#include <stdio.h>

void testFkt()
{
    int var1 = 0;
    static int var2 = 0;
    var1++;
    var2++;
    printf("var1: %i, var2: %i \n", var1, var2);
}

int main()
{
    testFkt();
    testFkt();
    testFkt();
}
```

Utskrift:

```
var1: 1, var2: 1
var1: 1, var2: 2
var1: 1, var2: 3
```

var2 initialiseras till noll vid programmets uppstart och behåller sitt värde mellan anrop.

var2 är statisk variabel som endast har lokal synlighet inom sitt scope – dvs här för testFkt().

Static variant 2 – vanligt exempel

```
#include <stdio.h>

void testFkt()
{
    static int bFirstTime = 1;
    if(bFirstTime) {
        // T ex allokera minne på heapen
        // eller förberäkna något.
        bFirstTime = 0;
    }
    … Gör huvudberäkningarna.
}

int main()
{
    testFkt();
}
```

T ex: rita gubbe, skriv ut resultat i ett fönster, gör avancerad AI-simulering, skicka data över nätverk.

Några fler qualifiers

- const
- inline
- static
- extern
- volatile
- (restrict)

```
const double pi = 3.14159265359; // variabeln kan inte ändras

x = 90 * pi / 180; // kan förberäknas redan vid kompileringen

// pekaren får nu inte ändra värde till annan adress
volatile unsigned char * const inport = (unsigned char*) 0x40021010;
```

```
static inline int square(int x)
{
    return x * x;
}

int main()
{
    int a = square(5);
}
```

Typer läses lättast från höger -> vänster

#define – find and replace på textnivå. Typedef funkar inte på samma sätt (men ofta ingen praktisk skillnad)

- **typedef** int postnr;
  typedef int gatunummer;
  - postnr x = 41501;
  - Gatunummer y = 3; // Observera att x och y nu har samma typ så man kan skriva x = y;

Angående typer: Läs från **höger till vänster**: (const finns sedan C89/90)

- unsigned char * const inport = (unsigned char*) 0x400; // inport är en constant pekare till en unsigned char
- unsigned char const * inport2 = (unsigned char*) 0x400; // inport är en pekare till en constant unsigned char
- const unsigned char * inport2 = (unsigned char*) 0x400; // inport är en pekare till en constant unsigned char
- char unsigned const * inport2 = (unsigned char*) 0x400; // inport är en pekare till en constant unsigned char
- char unsigned const * const inport3 = (unsigned char*) 0x400; // inport är en constant pekare till en constant unsigned
  // char

MEN… (som exempel på att typedef ej motsvarar find and replace)

- typedef unsigned char* port8ptr;
- const port8ptr p = (port8ptr)0x400;
- const unsigned char* p2 = (port8ptr)0x400;
- p2=0; // OK för p2 är pekare till const unsigned char
- p = 0; // ej OK för p är const-pekare till unsigned char
- “port8ptr const p3” är samma som const port8ptr p3”

Foo kan inte inline:as såhär (< gcc 2010)

```
// main.c
#include <stdio.h>
#include "foo.h"

int main()
{
      printf("x is %i", foo(0));
      return 0;
}
```

```
// foo.h
inline int foo(int x);
```

```
// foo.c
#include <stdlib.h>

inline int foo(int x)
{
      if( x == 0 ){
            int x = 4;
            return x;
      }

      return x;
}
```

Anledningen är att när kompilatorn kompilerar main.c finns inte definitionen av foo() tillgänglig – utan bara deklarationen. (Deklarationen ligger i foo.h. Definitionen ligger i foo.c.)

Leder i själva verket ofta till kompileringsfel.

Nu kan foo inline:as

```
// main.c
#include <stdio.h>
#include "foo.h"

int main()
{
     printf("x is %i", foo(0));
     return 0;
}
```

```
// foo.h
static inline int foo(int x)
{
     if( x == 0 ){
          int x = 4;
          return x;
     }

     return x;
}
```

`static` är ofta nödvändigt

c-fil
Inkluderar header-fil

header-fil

För main.c syns nu hela definitionen av foo() ☺.

Inline och h-filer

Bonus

```
// vecmath.h
#ifndef VECMATH_H
#define VECMATH_H

typedef struct {
   union {
      float v[2];
      struct { float x,y; };
   };
} Vec2f;

typedef struct {
   Vec2f min;
   Vec2f max;
} Box2f;  // box

static inline Box2f Add(Box2f * box, Vec2f v)      // Adds vector v to min and max of box
{
   Box2f res;
   res.min.x = box->min.x + v.x;
   res.min.y = box->min.y + v.y;
   res.max.x = box->max.x + v.x;
   res.max.y = box->max.y + v.y;
   return res;
}
#endif //VECMATH_H
```

inline innebär att man ber kompilatorn att stoppa in koden direkt i anropande funktion istället för att göra ett funktionsanrop. Då slipper man overhead för branching och aktiveringspost. För att kunna göra detta får ej inline-funktionen ligga i en annan .c-fil (eftersom den aldrig syns vid kompileringen utan först vid länkningen. Vissa kompilatorer klarar dock inline även vid länkningssteget.

Vissa kompilatorer kräver static framför inline, ty om inline misslyckas så kan funktionen Add() bli globalt **definierad** (dvs får skapad kod som är globalt synlig vid länknigen) för varje .c-fil som inkluderar vecmath.h, **om** det **inte** stod static. Static betyder att Add() bara får .c-fillokal synlighet. Utan static genereras länkfelet “multiple definition” om vecmath.h inkluderas av flera .c-filer.

Dynamisk minnesallokering

- `malloc()` Allokera minne
- `free()` Frigör minne

Funktionsprototyp via:

```
#include <stdlib.h>
```

Dynamisk minnesallokering

```
#include <stdlib.h>

char s1[] = "This is a long string. It is even more than one sentence.";

int main()
{
    char* p;

    // allokera minne dynamiskt
    p = (char*)malloc(sizeof(s1));

    // gör något med minnet som vi reserverat

    // frigör minnet
    free(p);
    return 0;
}
```

Antal bytes vi allokerar

OUT OF MEMORY
virtuellt adressutrymme uppdelat i pages som swappas mot hårddisk. Även detta kan ta slut -> krasch.

Minnesallokeringsexempel

```
// Kopiera från s1 till det allokerade minnet som pekas ut av p.
#include <stdio.h>
#include <conio.h>
char s1[] = "This is a long string. It is even more than one sentence.";
int main()
{
		char* p;
		int i;
		// allokera minne dynamiskt
		p = (char*)malloc(sizeof(s1));

		// gör något med minnet som vi reserverat
		for( i=0; i<sizeof(s1); i++)
				*(p+i) = s1[i];
		printf("%s", p);

		// frigör minnet
		free(p);

		return 0;
}
```

Minnesläckor

- En minnesläcka uppkommer om vi inte frigör det minne som vi allokerat med `malloc()`.
- Minnesläckor kan orsaka systemhaveri om minnet tar slut.
- Minnesläckan försvinner när programmet terminerar.

Hitta minnesläckor

- Man kan använda en minnesanalysator som t ex DrMemory (http://www.drmemory.org/)
- DrMemory ersätter standard biblioteket, och analyserar anrop till `malloc()` och `free()`.
- `Hittar även accesser till oinitierat minne`

Övningsuppgifter

1.

```
unsigned int a = 0x44332211;
unsigned char b = … a …; // b ska tilldelas värdet av 3:e byten i a
```

2. Hur gör du gVar synlig i main.c?

```
// main.c
int main()
{
    gVar = 5;
    return 0;
}
```

```
// foo.c
int  gVar = 1;
```

3. Hur hindrar du multipla inkluderingar för myfile.h (dvs gör include guards)?

```
// myfile.h

typedef struct {
    int v[2];
} myStruct;
```

Nästa Föreläsning:

Realtidsloop, grafikloop, fraktalt berg

C99, Dubbelpekare