Grundläggande C-programmering – del 5
Applikationsbyggnad/Spelprogrammering (realtidsstyrsystem) och Avancerad C

Ulf Assarsson

Läromoment:

- Realtidsloop
- Överkurs:
  - grafikloop, fraktalt berg
  - C99
  - obskyra C-konstruktioner
  - Dubbelpekare,

Kopplat till:

- Lab 5 - spelprogrammering

Föregående lektion

- Synlighet – static, #extern, (inline), #if/#ifdef, #include guards,

```
void testFkt()
{
    static int timesVisited = 0;
    timesVisited++;
}
```

```
extern int g_var;
…
g_var = 5;
```

```
// I annan .c-fil:
int g_var;
```

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

extern OBJECT player;

void movePlayer(Vec2i v);

#endif //PLAYER_H
```

- enum, union, little/big endian

```
enum day {monday=1, tuesday, wednesday, thursday, friday, saturday, sunday};
enum day today;
today = wednesday; // == (int)3
```

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

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

- Dynamisk minnesallokering (malloc/free)

```
int* p;
p = (int*)malloc( 1000 * sizeof(int));
…
free(p); // när man vill deallokera
```

Överkurs

Kodningskonventioner

- Aktiveringspost (stack frame)
- Prolog
- Epilog

Varför kodningskonventioner?

- Regler för
  - Funktionsargument.
  - Returvärde.
  - (Hur lokala variabler skapas.)
- Möjliggör
  - Separat kompilering av olika .c filer
  - Mix av assembler och C.

Detaljerna styrs av konventioner:

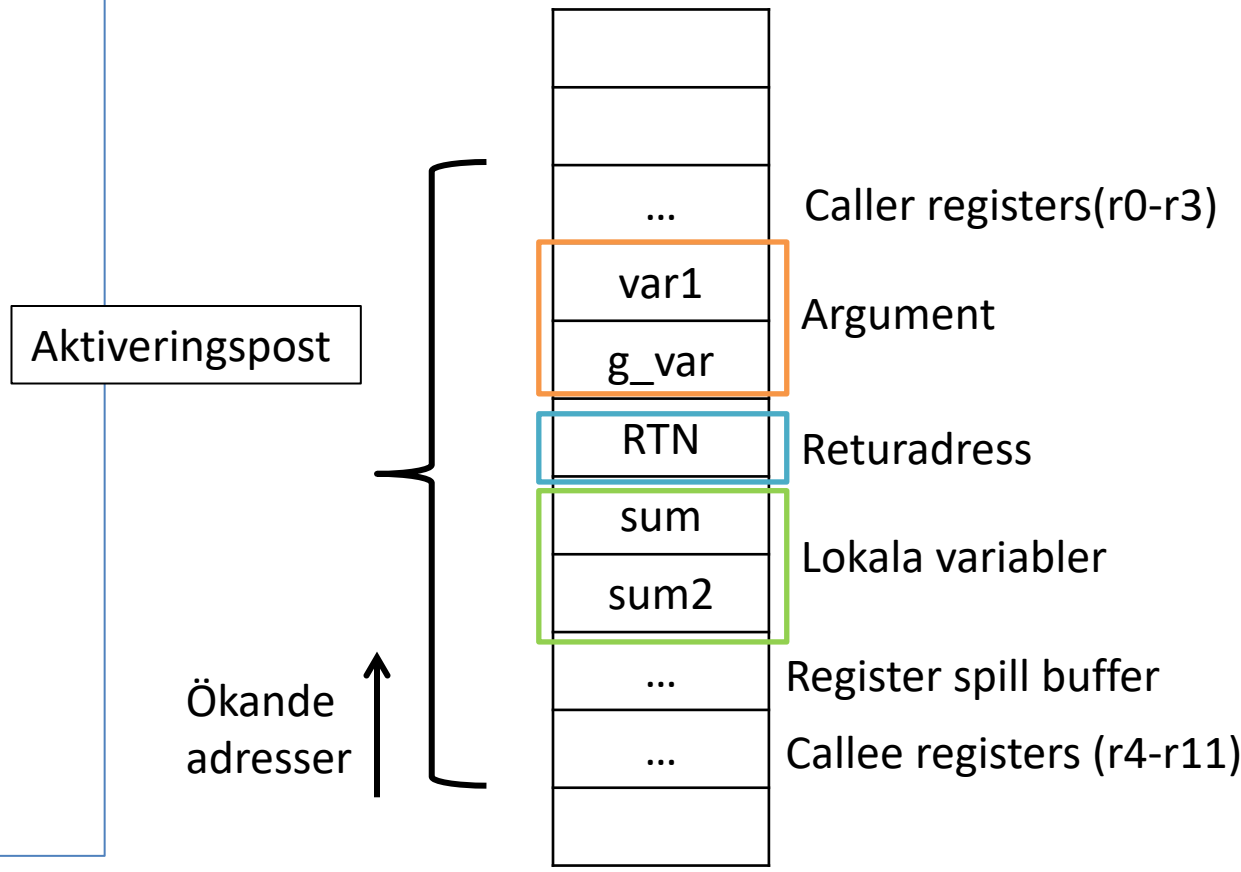
- C har övergripande regler för parameteröverföring (höger till vänster).
  Resten är processorspecifikt och styrs av:
  - "application binary interface" (ABI) – calling conventions
  - Konventionen används av kompilatorn eller dig som assemblerprogrammerare

Funktionsanropskonventioner
(calling conventions)

- *Parametrar, två metoder kombineras*
  - Via register: snabbt, effektivt och enkelt,
    Nackdel: finns bara begränsat antal register
  - Via stacken: generellt
- *Returvärden*
  - Via register:för enkla datatyper som ryms i processorns register. T ex R0 (plus ev. R1, om 64-bits värde)
  - Via stacken: sammansatta datatyper (poster och fält)

Aktiveringspost / Stack frame – ett generaliserat exempel (ej exakt för ARM)

```
int myAdd(int x, int y)
{
    // lokala variabler
    int sum, sum2;
    sum = x+y;
    return sum;
}


var2 = myAdd(g_var, var1);
```



Men: ARM-Cortex M4: använder ofta LR (r14), the link register, istället för att lägga returadressen på stacken

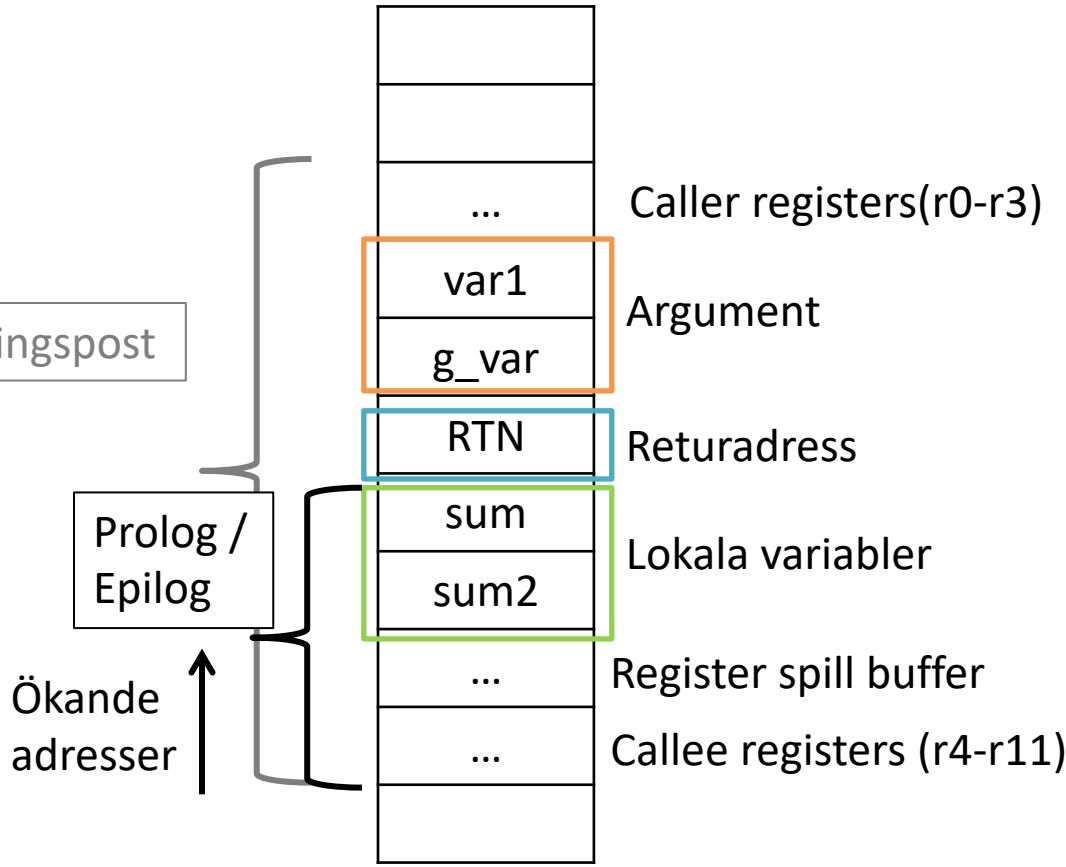
Aktiveringspost / Stack frame – ett generaliserat exempel (ej exakt för ARM)

```
int myAdd(int x, int y)
{
    // lokala variabler
    int sum, sum2;
    sum = x+y;
    return sum;
}


var2 = myAdd(g_var, var1);
```

Prolog

Epilog

Aktiveringspost



Men: ARM-Cortex M4: använder ofta LR (r14), the link register, istället för att lägga returadressen på stacken

Aktiveringspost / Stack frame – med struct som returparameter

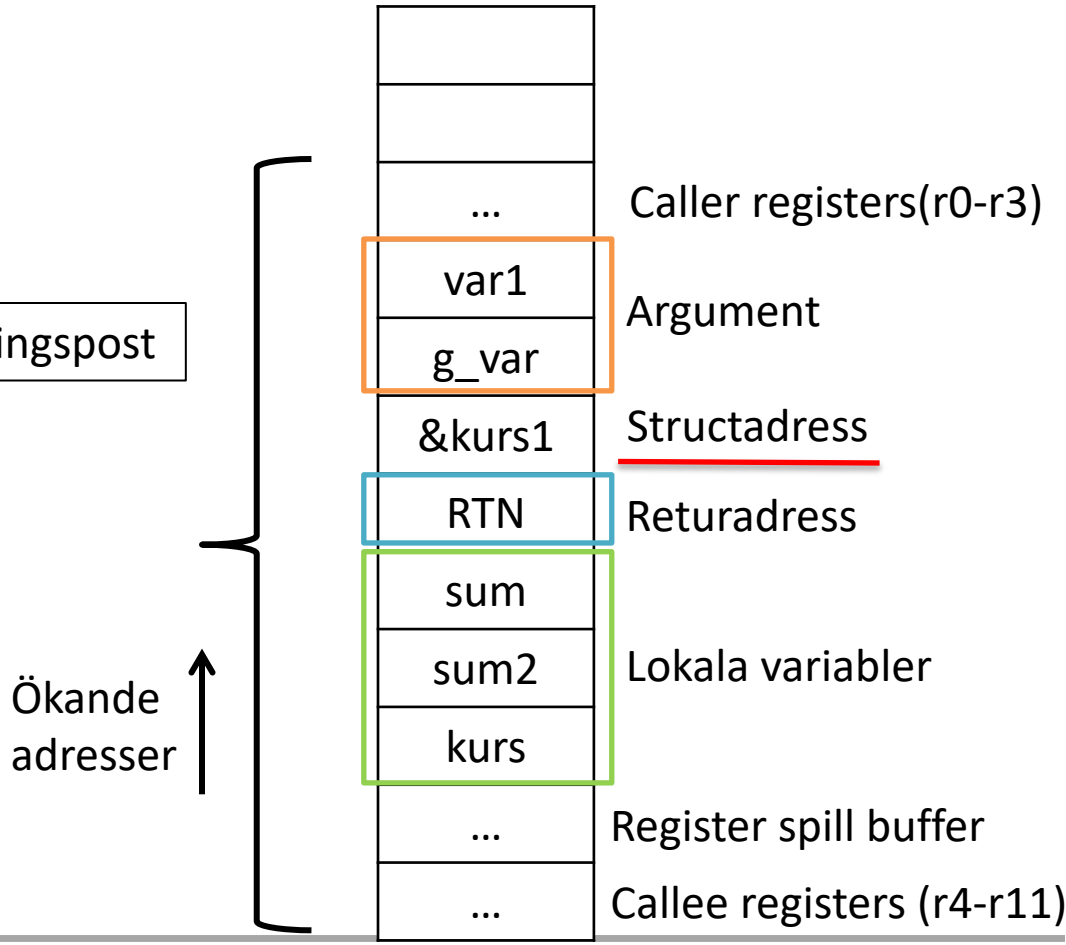
Returstructadressen pushas som en hemlig 0:e inputparameter.

```
struct Course myFkt(int x, int y)
{
    // lokala variabler
    int sum, sum2;
    struct Course kurs;
    …
    kurs.name = "MOP";
    return kurs;
}

struct Course kurs1;
kurs1 = myFkt(g_var, var1);
```

Prolog

Epilog

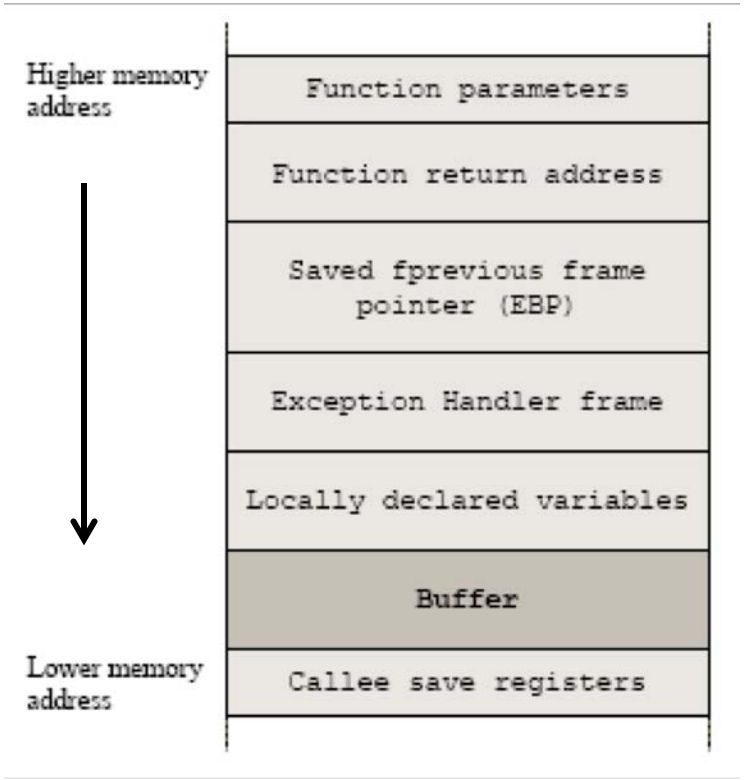
Aktiveringspost



Prolog och Epilog

- Prologen av en funktion skapar utrymmet för lokala variabler etc.
  - Görs genom att flytta (dekrementera) SP
- Epilogen av en funktion återlämnar minnet för lokala variabler etc.
  - Görs genom motsvarande tillbakaflyttning (inkrementering) av stackpekaren.

Aktiveringspost hos x86



Säkerhetsrisk…

Generaliserat recept för ett funktionsanrop

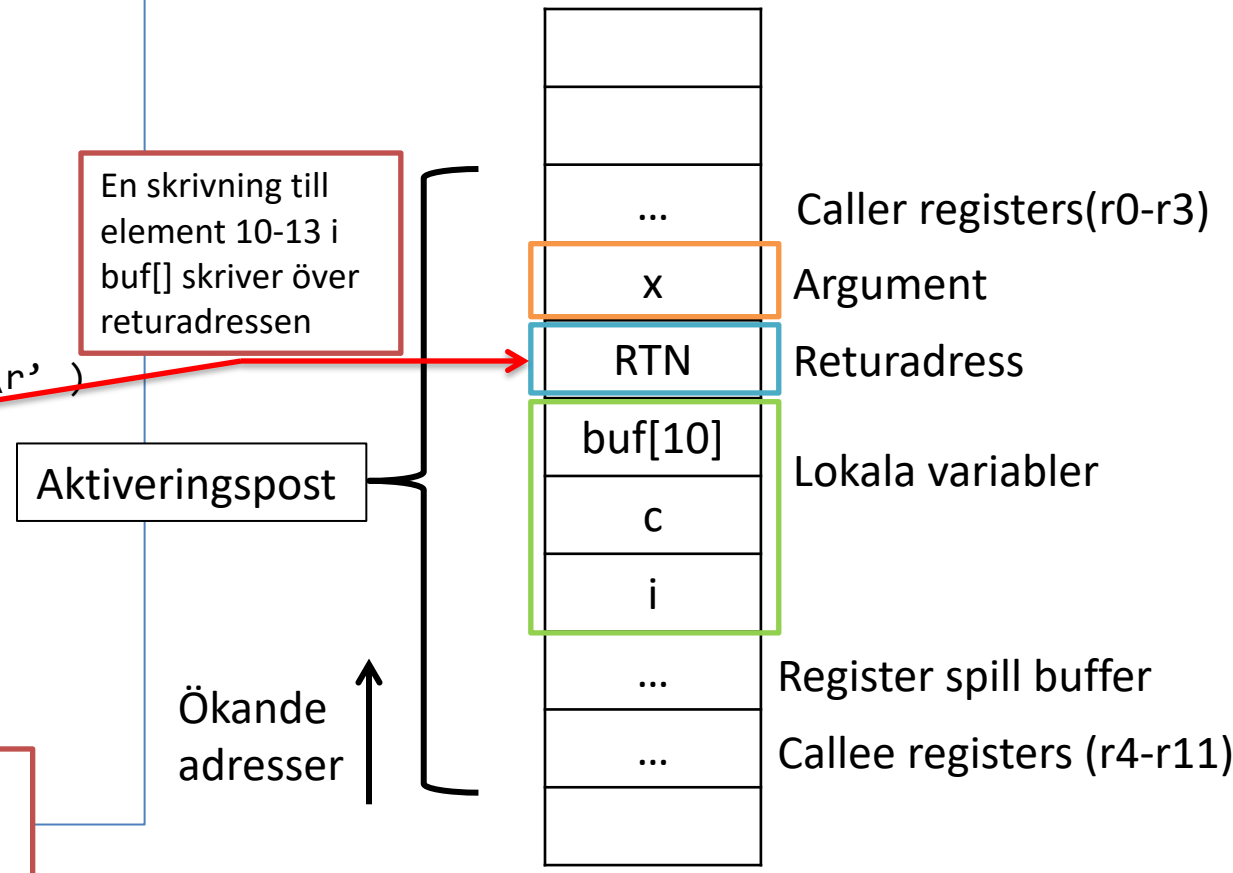
1. Pusha caller save registers (för ARM cortex-m4: r0-r3) på stacken
2. Pusha argumenten på stacken (och/eller använd register r0-r3).
3. ”JSR” (dvs hoppa till subrutin):
   - *BL*
   - eller: *PUSH {LR}, BL … POP {LR}* (dvs pusha återhoppsadressen på stacken).
4. Prolog: skapar utrymme för lokala variabler etc.
   - och spara callee save registers (r4-r11)
5. { Funktionskroppen utförs }
6. Lägg returvärde i rätt register eller kopiera direkt till rätt structadress.
7. Epilog: återlämnar utrymme för lokala variabler etc.
   - och återställ Callee save registers (r4-r11)
8. ”RTS” (returnera från subrutin):
   - BX LR
   - eller POP {PC} (pop till programräknaren PC om returadressen ligger där stackprekaren pekar)
9. Återställ stacken till tillståndet innan argumenten pushades.
10. Poppa caller save registers (r0-r3) från stacken

Exekvera skadlig kod via stacken…

```
void osäkerKod(int x)
{
    // lokala variabler
    char buf[10];
    char c;
    int  i=0;

    // unsafe input
    while ( (c=getch()) != '\r' )
        buf[i++] = c;
    buf[i] = '\0';
    …
    return;
}
```

En skrivning till element 10-13 i buf[] skriver över returadressen

Aktiveringspost



(OBS. Ofta tillåter dock inte ett modernt operativsystem att exekvera programkod som ligger på stacken.)

För Windows, CodeLite, 64-bits gcc

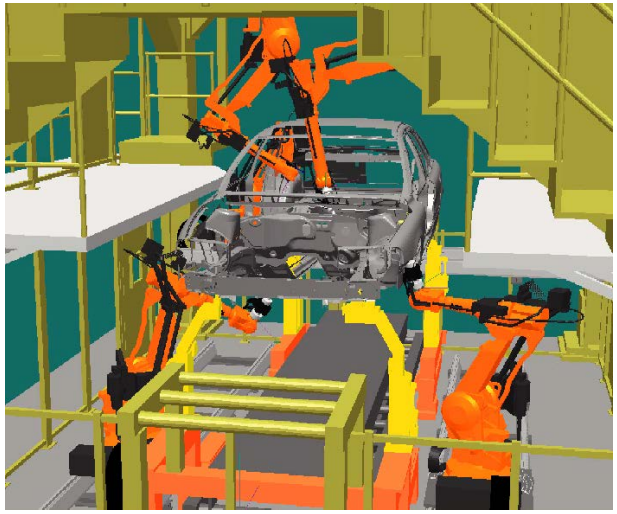
```
#include <stdio.h>
#include <conio.h>

void some_func()
{
   // Desired function to be launced
   printf("\nOutputting $1000. \n Take your money and your receipt!\n");
}

int main(int argc, char **argv)
{
   printf("How a user can launch any desired function via an unsafe text-input function.\n");
   input();
   return 0;
}

void input ()
{
   char buf[10]; // begränsad buffer, på stacken
   char c;
   int  i=0;

   printf("Input text: ");    // typical unsafe input
   while ( (c=getch()) != '\r' ) // obegränsad input
      buf[i++] = c;  // kan skriva över returadressen
   buf[i] = '\0';      // end of typical unsafe input

   *(long long *)(&buf[24]) = (long long) &some_func;}
}
```

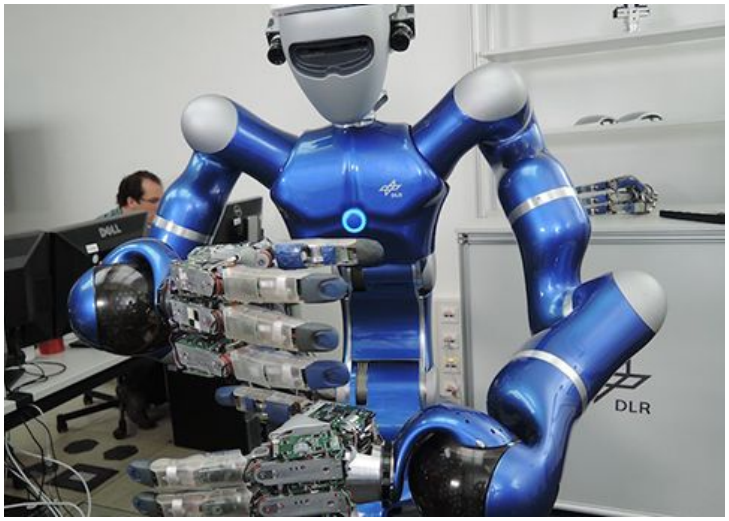
Här kan man mata in egen returadress inkl. egen kod på stacken i värsta fall.

```
// För att exemplifiera matar vi in adressen till some_func() manuellt:
// After analysis, the return PC is 24 bytes after buf[0]
// (20 bytes for 32-bits applications (on Windows, Intel, gcc))
```

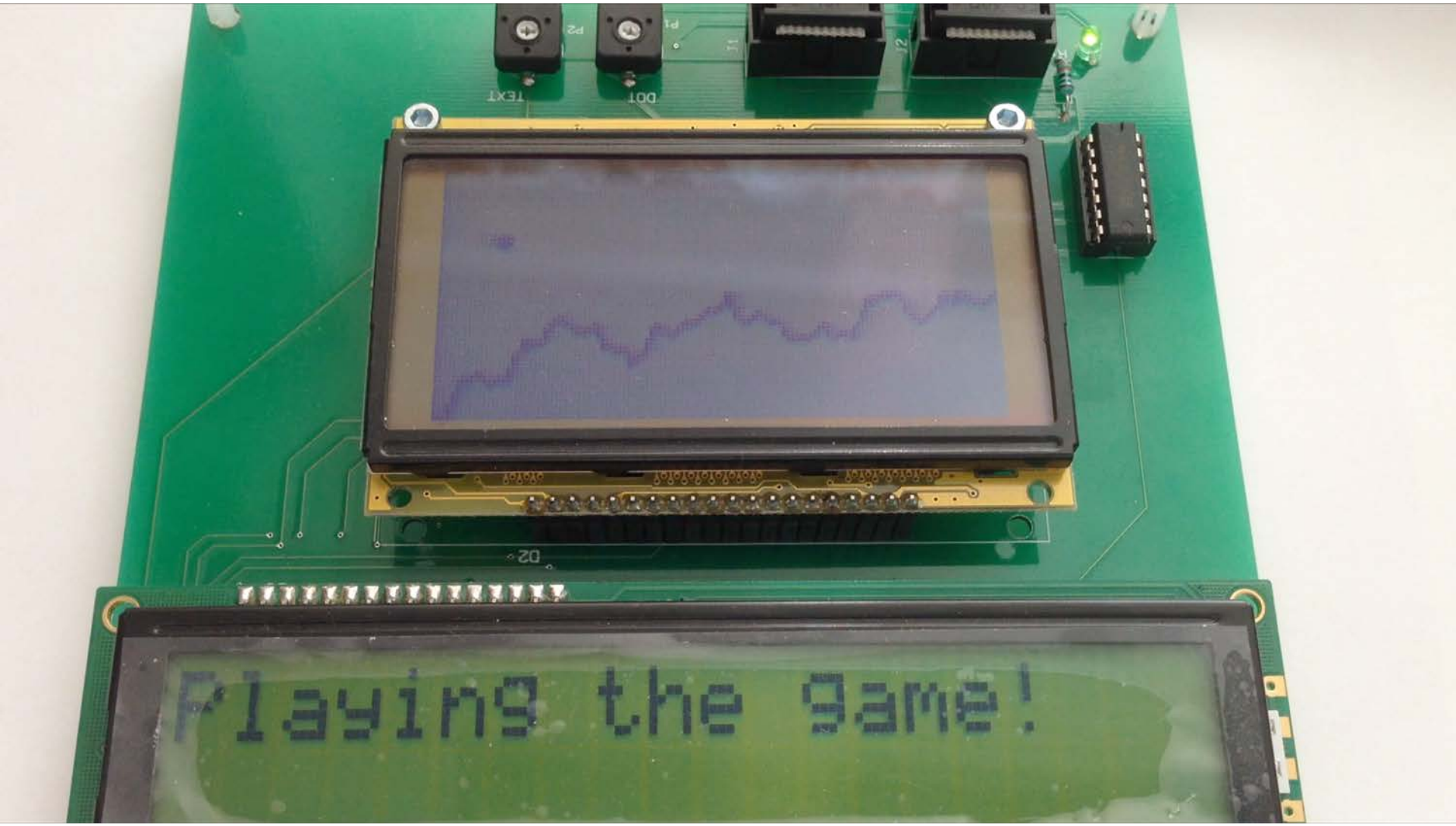
Spelprogrammering (realtidsstyrsystem)

- Realtidsspel är ofta avancerade exempel på styrsystem i realtid.
  - Jfr animera ABB-robot med att animera avatarer eller andra spelobjekt.

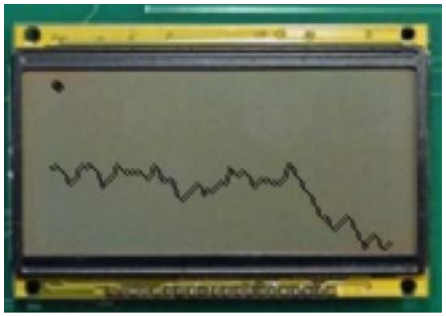




Ett enkelt spel…



Game loop



En typisk game loop.

```
int main(int argc, char **argv)
{
   appInit();                      // initialize GPIO_E
   graphic_initalize();            // initialize the lcd/ascii-display
   graphic_clearScreen();
   clearBuffers();                 // clear front/backbuffer
   bool gameover = false;
   while( !gameover )    {  // Game loop (dvs realtidsloopen)
      clearBuffer(0);
      collisionDetection(); // sets flags on the objects
      updatePlayer();       // user input
      updateObjects();      // moves and/or kills objects based on the flags
      drawObjects();        // draws the objects to back buffer
      swapBuffers();        // draw backbuffer and swap back/frontbuffer.
      delay_milli(40);      // ~25 frames/sec
   }
}
```

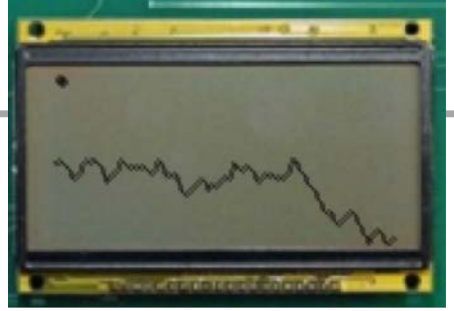
Game loop

```
OBJECT* objects[] = {&player, osv…};
unsigned int nObjects = 1;

void updateObjects() {
   for(int i=0; i<nObjects; i++)
      objects[i]->move(objects[i]);
}

void drawObjects() {
   for(int i=0; i<nObjects; i++)
      objects[i]->draw(objects[i]);
}

void updatePlayer() {
   switch( tstchar() ) { // tstchar() checkar input via USART-porten
      case '6':  player.set_speed( &player, 2,  0); break;
      case '4':  player.set_speed( &player, -2, 0); break;
      case '8':  player.set_speed( &player, 0, -2); break;
      case '2':  player.set_speed( &player, 0,  2); break;
   }
}
```

```
void collisionDetection()
{
    ; // insert your own code
}
```

```
static OBJECT player = {
  &ball_g,  // geometri för en boll
  0,0,      // initiala riktningskoordinater
  1,1,      // initial startposition
  draw_object,    // funktionspekare
  clear_object,
  move_object,
  set_object_speed
};
```

Game loop



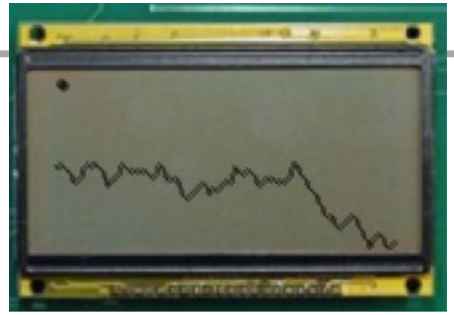
En realistisk grafikloop.

```
unsigned char frameBuffer0[1024], frameBuffer1[1024];
unsigned char *frontBuffer = frameBuffer0;
unsigned char *backBuffer = frameBuffer1;

void clearBuffer(unsigned char val) {
    for (int i=0; i<1024; i++)
        backBuffer[i] = val;
}
void clearBuffers() {
    for (int i=0; i<1024; i++)
        backBuffer[i] = frontBuffer[i] = 0;
}

void swapBuffers() {
    graphic_drawScreen();
    unsigned char* tmp = frontBuffer; // swap front/backbuffers
    frontBuffer = backBuffer;
    backBuffer = tmp;
}
```

```
int main(int argc, char **argv)
{
   appInit();
   graphic_initalize();
   graphic_clearScreen();
   clearBuffers();

   bool gameover = false;

   while( !gameover ) {
      clearBuffer(0);
      collisionDetection();
      updatePlayer();
      updateObjects();
      drawObjects();
      swapBuffers();
      delay_milli(40);
   }
}
```

Motivering:
Att skriva ut på displayen är långsamt. Har man 100-tals objekt är det snabbare att istället kopiera ut en hel skärm från en framebuffer och dessutom slippa `graphic_clearScreen()`.

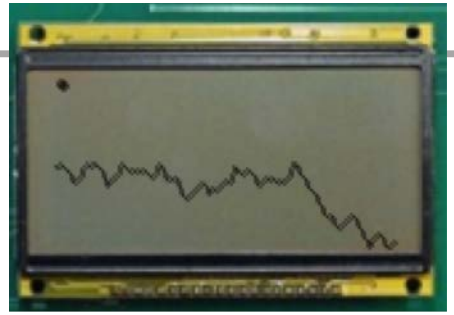
Game loop



En realistisk grafikloop.

```
void graphic_drawScreen(void)
{
   unsigned int k = 0;
   bool bUpdateAddr = true;
   for(uint8 c=0; c<2; c++) {                          // loop over both controllers (the two displays)
      uint8 controller = (c == 0) ? B_CS1 : B_CS2;
      for(uint8 j = 0; j < 8; j++) {                   // loop over pages
         graphic_writeCommand(LCD_SET_PAGE | j, controller );
         graphic_writeCommand(LCD_SET_ADD | 0, controller);
         for(uint8 i = 0; i <= 63; i++, k++) {         // loop over addresses
            // update display only where it is different from last frame (tst diff of front/backbuffer)
            if( backBuffer[k] != frontBuffer[k] ) {
               if(bUpdateAddr )
                  graphic_writeCommand(LCD_SET_ADD | i, controller);
               graphic_writeData(backBuffer[k],  controller);
               bUpdateAddr = false; // Display hardware auto-increments the address per write
            } else
                bUpdateAddr = true; // No write -> we need to update the x-address next write
            }
         }
      }
   }
}
```

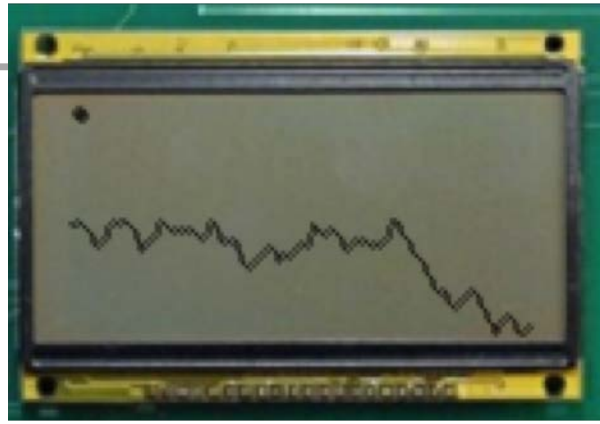
Game loop



Vi får då även uppdatera pixel().

```
void pixel( int x, int y, int set )
{
   if (!set) return;

   if( (x > 127 )  || (x < 0) || (y > 63) || (y < 0) )
      return;

   unsigned char mask = 1 << (y%8);

   int index = 0;

   if(x>=64) {
      x -= 64;
      index = 512;
   }
   index += x + (y/8)*64;

   backBuffer[index] |= mask;
}
```

Fractal mountains

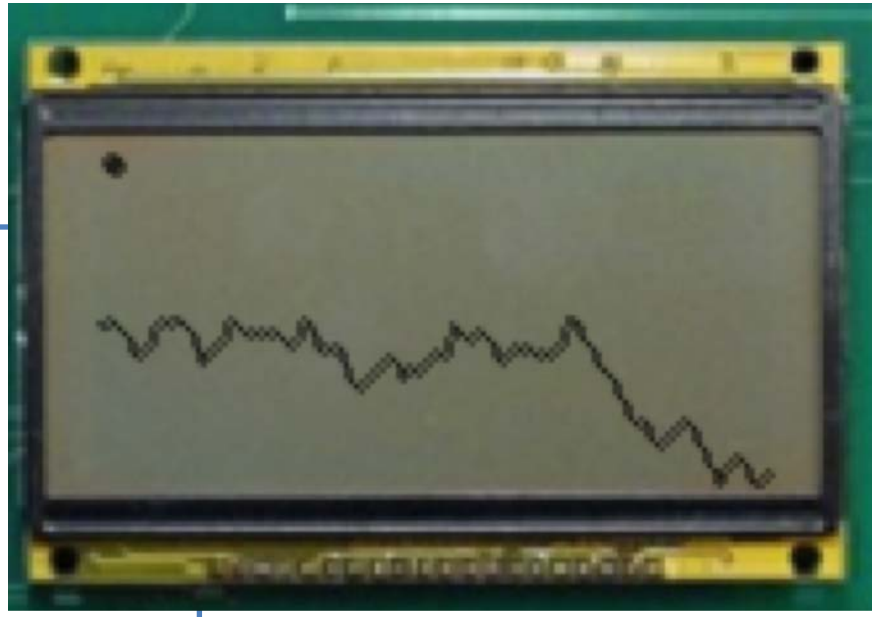


```
POBJECT objects[] = {&player, &fmountain};
unsigned int nObjects = 2;
```

```
static OBJECT fmountain = {
  0,          // geometri behöver vi ej
  0,0,        // initiala riktningskoordinater
  0,0,        // initial startposition
  drawMountain,    // funktionspekare
  0,                // unused
  move_object,        // dummy
  set_object_speed    // dummy
};
```

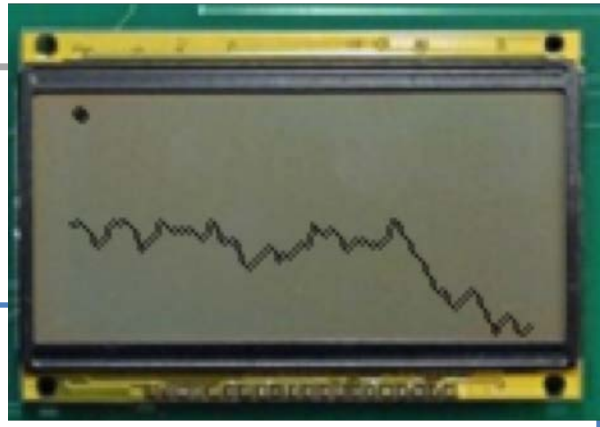
```
static OBJECT player = {
  &ball_g,  // geometri för en boll
  0,0,        // initiala riktningskoordinater
  1,1,        // initial startposition
  draw_object,    // funktionspekare
  clear_object,
  move_object,
  set_object_speed
};
```

Fractal mountains



```
static unsigned char y_values[128];

void drawMountain() {
   static bool bFirstTime = true;
   if(bFirstTime) {
      bFirstTime = false;
      for(uint8 x=0; x<128; x++)
         y_values[x] = fractal();
   } else {
      // shift mountain to the left
      for(uint8 x=0; x<127; x++)
         y_values[x] = y_values[x+1];
      y_values[127] = fractal();
   }
   // anropa pixel för alla x=[0,127]
   for(uint8 x=0; x<127; x++)
      pixel(x, y_values[x], 1);
}
```

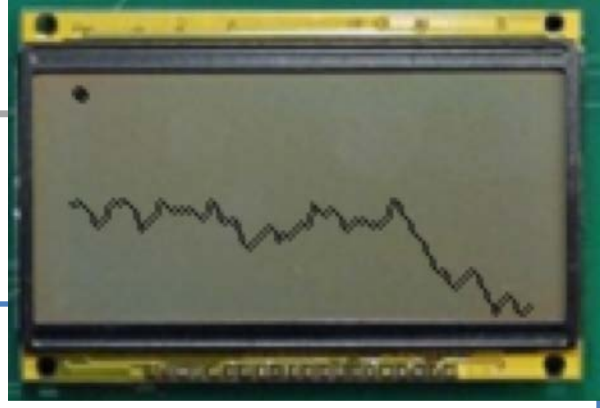
Fractal mountains



```
#define RAND_MAX 32767
static unsigned int next = 1;
static unsigned int rand(void)
{
   next = next * 1103515245 + 12345;
   return (unsigned int)(next/65536) % 32768;
}
static void srand(unsigned int seed) { next = seed; }
```

Rand() finns i stdlib.h, men vi saknar ju alla standardbibliotek för MD407:an, så vi måste implementera själva, t ex enligt ovan.

Fractal mountains



```
static unsigned char fractal()
{
   // f = noise(t)*A + noise(0.5t)*2A + noise(0.25t)*4A...
   static int noise[] = {0,0,0};  // room for 3 octaves
   static int t = 0;
   static int f = RAND_MAX / 2;
   const int half_max = RAND_MAX/2;
   noise[0] = (rand() > half_max) ? 1 : -1; // update each time
   if( (t%2) == 1)
      noise[1] = (rand() > half_max) ? 1 : -1; // update every 2nd time
   if( (t%4) == 3)
      noise[2] = (rand() > half_max) ? 1 : -1; // update every 4th time
   // sum octaves
   int val = 0;
   for(int i=0; i<3; i++)
      val += noise[i];
   f += val; // update our static non-bounced fractal function
   t = (t+1) % 4;
   val = f % 64;    // return a bounced value between 32 + [0-32]
   val = (val > 31) ? 63-val : val;
   return val + 32;
}
```

Non-Standard C Extensions

C – non-standard extensions

- Följande finns som extensions till C
  - Stöds typiskt inte alls i C++
  - Dessa stöds i C99 men typiskt ej i C89 / C11 (möjligen som optional för kompilaror-tillverkaren).

C – nested/local functions are compiler-optional extensions to C

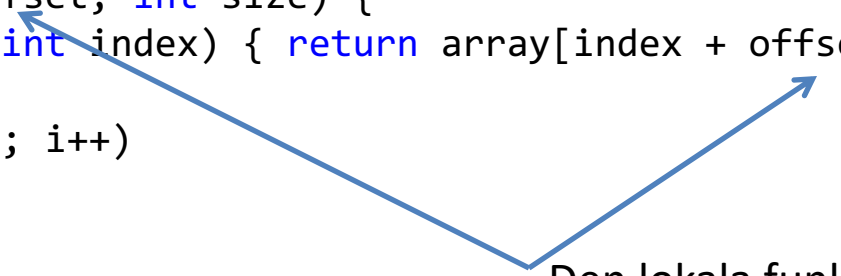
VC++ (dvs Visual Studio C++) och GNU C++ stödjer inte lokala funktioner. Det gör däremot ARM-gcc och MinGW-gcc.

```
double fkn(double a, double b)
{
  double square (double z) { return z * z; }

  return square (a) + square (b);
}
```

C – nested/local functions are compiler-optional extensions to C

VC++ (dvs Visual Studio C++) och GNU C++ stödjer inte lokala funktioner
Det gör däremot ARM-gcc och MinGW-gcc.

```
void fkn(int *array, int offset, int size) {
   int access (int *array, int index) { return array[index + offset]; }

   for (int i = 0; i < size; i++)
      access (array, i);
}
```

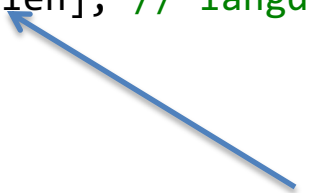


Den lokala funktionen access() har t.o.m. tillgång till omgivande scope's hittills deklarerade variabler. Här "offset" i fkn()

C99 – variable-length arrays

(Optional in C11)

```
void fkn(int len)
{
    …
    char str1[10];  // längd känd i compile time.
    char str2[len]; // längd ej känd i compile time men i run time.
    …
}
```



str2 är en variable-length array (C99)

C99 – struct initiation med {.medlem, }

Initiering med { .medlem = ... } tillåter oss att endast initiera valfria medlemmar.

Exempel:

```
struct Tst {
    int a;
    char b;
};

struct Tst x1 = { .b = 'z' }; // ofullständig initiering
struct Tst x2 = { .b = 'z', .a = 5 }; // initiering i valfri ordning
```

Luriga uttryck

Möjligt att göra – men gör inte såhär!

☺

C – luriga uttryck (överkurs)

Godtyckliga expressions.

```
for (expr/dekl; expr; expr)
{
    statement;
}

if( expr )
{
   …
}
```

Examples of statements:
`if, while, do/while, for, switch, expr`

Deklaration av lokala variabler:
`int a, b=0;`

Examples of expressions:

```
x = y + 3;
x++;
x = y = 0;          // Both x and y are initialized to 0
proc( arg1, arg2 );// Function call
y = z = f(x) + 3;  // A function-call expression

expr, expr;  // list of expressions. Conditional value
             // of the expression is the result of the
             // last expression.
```

C – luriga uttryck (överkurs)

```
for(expr/dekl; expr; expr)
{
    statement;
}
```

Godtyckligt expression/dekl. Dock typiskt en vanlig deklaration och initiering av loopvariabeln.

```
for (int a=1, b=0; b<5, a++, f(a), a<3; b++, a += (b>a) ? 1 : -1)
{
    a++, b++;
}
```

Godtyckligt expression.

OBS – vilkorets värde endast lika med sista uttrycket a<3 (så b<5 har här ingen effekt).

```
if(a=0, b++, a = (b == c) )
{
    …
}
```

C – luriga uttryck (överkurs)

Ett till knäppt exempel

```
void f(int b, int c)
{
   printf("%d, %d", b, c);
}

void main()
{
   int a = 0;
   f( (++a, ++a, ++a), (++a, ++a));
}
```

Kommaseparerade expressions evalueras vä -> hö.
Så även oftast för inputparametrar (men ospecificerat).
Värdet för en lista av expr är värdet av sista uttrycket.

Expr för 1:a parametern. Expr för 2:a parametern.

Vad skrivs ut?
Svar: 3, 5

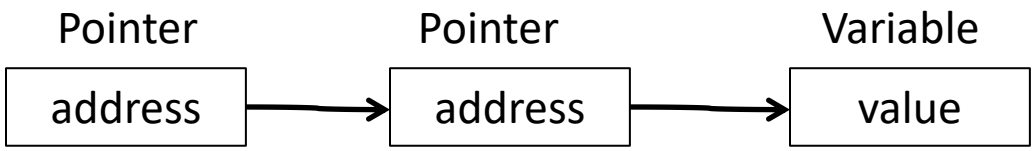
Men vad skrivs ut om man ändrar ++a till a++ ?

Dubbelpekare (om vi hinner)

(pekare till pekare)

Pekare till pekare

```
char a;
char *p = &a;
char **pp = &p;
```




```
// Exempel:

int main()
{
  int a = 5;
  int*     pa = &a;          // pekare
  int**   ppa = &pa;         // dubbelpekare
  int*** pppa = &ppa;        // trippelpekare

  ***pppa = 3;
  …

}
```

Pekare till pekare (dubbelpekare)

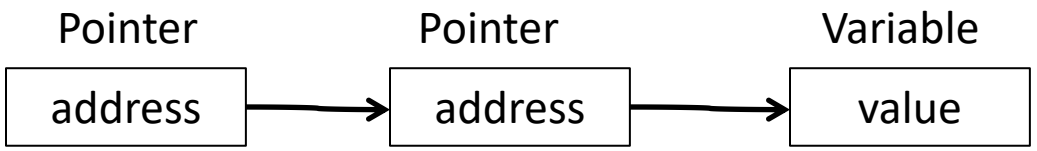
```c
#include <stdio.h>

void main()
{
        int a = 5;
        int* pa = &a;
        int** ppa = &pa; // ppa är dubbelpekare (pekare till pekare till int)

        printf("%i\n", **ppa);
}
```

Pekare till pekare (dubbelpekare)

```
#include <stdio.h>

char* s1 = "Emilia";  // variabeln s1 är en variabel som går att ändra, och
                      // vid start tilldelas värdet av adressen till 'E':
char s2[] = "Emilia"; // värdet på s2 känt vid compile time. s2 är konstant, dvs
                      // ingen variabel som går att ändra. Är adressen till 'E'.
int main()
{
        char* p = s1;  // == &(s1[0])
        char** pp = &p;
        **pp = 'A';   // ändrar s1[0] till 'A'
        printf("%s\n", p); // printar "Amilia"

        return 0;
}
```

Pekare till pekare

```
char a;
char *p = &a;
char **pp = &p;
```




```
// 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;
  allocateEnemyArray(&pEnemies, 100);
  …
  free(pEnemies);
}
```

pp är av typ dubbelpekare.
pp pekar på pEnemies som är av typ pekare.
*pp =… ändrar värdet för variabeln pEnemies.

Ska funktionen kunna uppdatera argumentet måste vi skicka in adressen för argumentet (istället för värdet på argumentet). Eftersom pEnemies är en pekare så är adressen till pEnemies av typen dubbelpekare (pekare till pekare till struct Enemy).

Dubbelpekare. Övning

```
void main()
{
    char s1[] = "Emilia";
    char **pp, *p;


   -- MODIFIERA s1 via dubbelpekaren pp---


    printf("s1 = %s", s1);

}
```

Dubbelpekare. Lösning

```
int main()
{
    char s1[] = "Emilia";
    char **pp, *p;
    p = &s1[0];       // p = adressen till 'E'

    // eller
    p = s1;           // dvs p pekar på arrayen s1.


    pp = &p;          // pp pekar på pekaren p
    **pp = 'A’        // dubbel avreferering av "pp som pekar på p som pekar på 'E'".
    // eller
    (*pp)[0] = 'A';   // *pp pekar på p. Så det blir: ”p[0]” = 'E'

    *(*pp + 2) = 'e'; // *(p + 2). Dvs innehållet i (p + 2) tilldelas 'e’.


    printf("s1 = %s", s1);
    return 0;
}
```

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 'R'.
int main()
{
          printf("Kul med pekare\n");

          char **pp, *p = s1;
          pp = &p;
          printf("p = %s\n", p);  // p == s1, så ekvivalent med att skicka s1 som argument.
          printf("*pp = %s\n", *pp);

          *pp = s2;         // *pp ekvivalent med p. Ändrar p till s2
          printf("p = %s\n", p);

          *p = 'T';         // (p == s2). *s2 ekv. med s2[0]. Ändrar s2[0] till 'T'
          **pp = 'J';       // *pp ekv med p. p==s2. ->  *s2 = 'J'. Ändrar s2[0] till 'J'
          *(*pp+2) = 'k';   // ändrar s2[2] till 'k'
          printf("%s\n", p);

          (*pp)[0] = 'P';   // *pp ekv med p (== s2). s2[0]. Ändrar s2[0] till 'P'
          printf("%s\n", p);

          return 0;
}
```

Emscripten – C web kompilator

C-kompilator till WebAssembly (eller javascript)

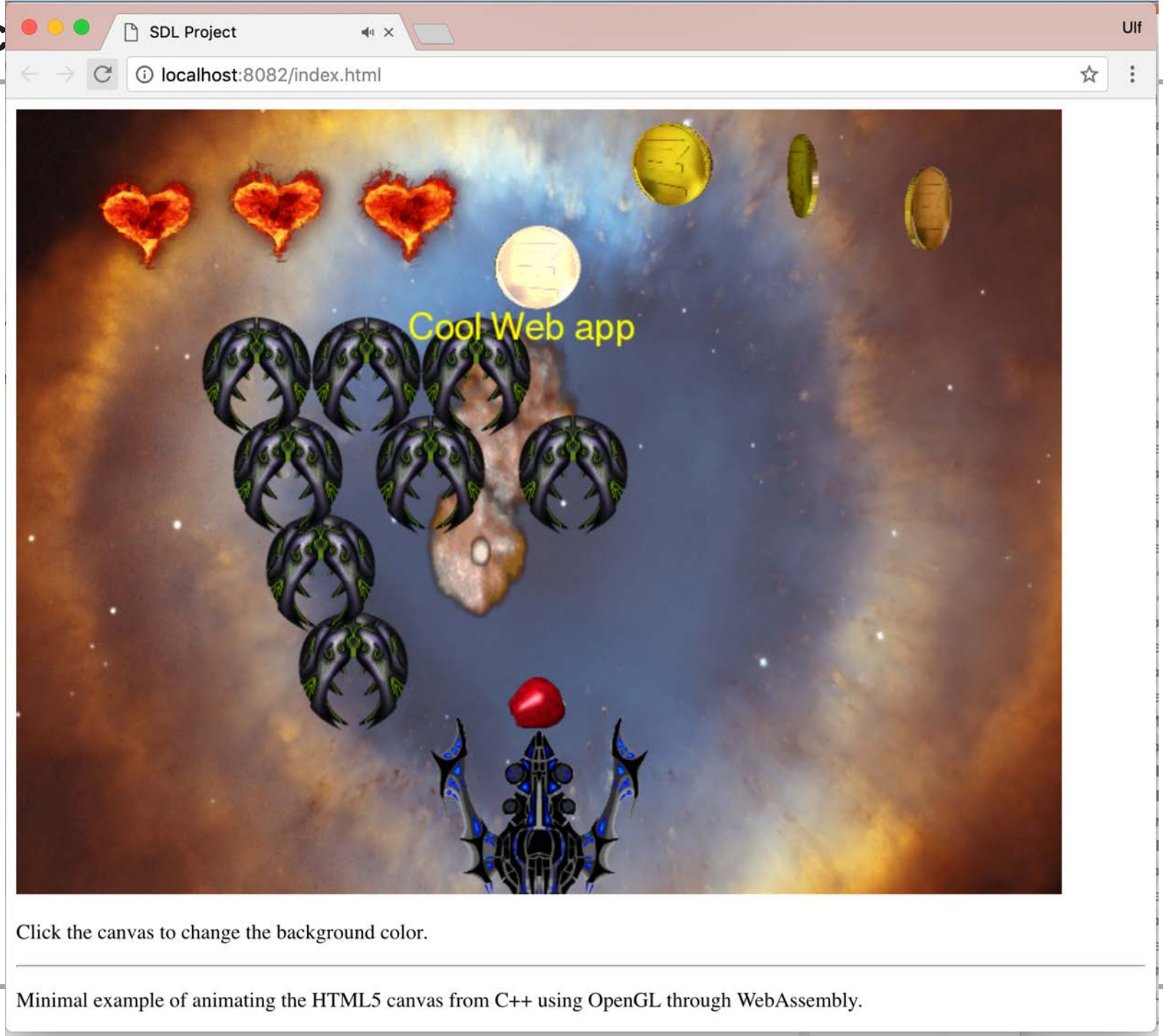
- Programmets input/output fungerar genom att:
  - Kapsla SDL2 för mus, tangentbord, gamepad, joystick, force feedback,…
  - Text via javascript-konsollen i browsern.
  - Filesystem: lagrar kopia av angiven filkatalog lokalt i RAM minnet

Med endast c:a 10 raders kodändringar kan vi kompilera hemuppgifterna till webbapplikation:

- Installera Emscripten version "sdk-tag-1.38.26-64bit"
- I CodeLite, lägg till *emcc* på samma sätt som *gcc*-kompilatorn
- Kompilera med emcc istället för gcc. (Sätt rätt flaggor... Komplicerat...)
- Sätt CodeLite->run att starta Chrome för genererad index.html

Se kommande websida för *hemuppgifterna*, *"C and WebAssembly"*

- Kommer innehålla en och samma CodeLite-projektfil för Windows/Mac/(Linux)/Emscripten.
  - Dvs samma projekt och kodbas kan kompileras mot native (Win/MacOS/Linux) eller webben med C


SDL Project
Ulf
localhost:8082/index.html
Cool Web app
Click the canvas to change the background color.
Minimal example of animating the HTML5 canvas from C++ using OpenGL through WebAssembly.