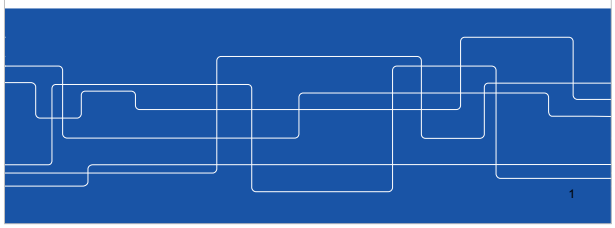
Föreläsning 12

Introduktion till C-programmering



Typer

- Python: dynamiskt och hårt typat språk
- C: *statiskt* och hårt typat språk

Typer (enkla):

Typ	Beskrivning	Bitar	Siffror	Exempel
int	heltal	32	9	17
float	flyttal (single prec)	32	6	3.14
double	flyttal (double prec)	64	15	3.14
char	tecken	8	(1)	'a'

Hello world

```
#include <stdio.h>

int main (int argc, char *argv[])
{
  printf (“Hello world!\n”);
  return 0;
}
```

1.Vi *definierar* ovan funktionen main.

2.Vi måste *deklarera* både returtyp (int) och argumenttyper (int, char *[]).

3.printf (...) anropar utskriftsfunktionen

4.#include <stdio.h> deklarerar bla printf (“gör den känd”)

Variabeldeklaration, tilldelning

```
<datatypen> <variabelnamn> [, <variabelnamn>, ...];
t.ex:
char c;
unsigned int age;
float weight, height;
/* deklaration och tilldelning på samma gång */
double length = 0.5;
```

Kompilering

- Programmets text kallas “källkod”
- Källkoden måste översättas till exekverbar kod (som datorns mikroprocessor förstår)
- Vi lägger källkoden till Hello world i filen hello.c och använder kompilatorn (översättaren) gcc:

  gcc -o hello hello.c
- Vi kan nu köra programmet:

  ./hello

Logiska uttryck

- C har ingen särskild typ för sanningsvärden (Python har True och False), utan att 0 betraktas som falskt och alla andra värden som sant.
- Relationsoperatorer:
  `> , < , >= , <= , == , !=`
- Logiska operatorer:
  `!  logisk negering`
  `&& logiskt och`
  `|| logiskt eller`

Logiskt uttryck exempel

```
m = (x >= 1.0 && x <= 2.00);
```

Variabeln m får värdet 1 om uttrycket är sant och 0 om det är falskt.

```
n = !(x < 10 || x > 21);
```

Variabeln n får värdet 1 om variabeln x t.ex har värdet 15.

Kommentar och block

- Kommentarer

```
/* detta är en rads kommentar */

/* detta är flera
   raders kommentar */
```

- Block

```
{
  <deklarationer och satser>
}
```

Ett block räknas som en sats!

Implicit typkonvertering

Typer konverteras i uttryck. Om någon av operanderna är av en "smalare" typ än den andra, konverteras denna till den "bredare" typen:

```
int n = 2;
float x;
x = n + 3.4;
n = x;
x = n = 2.4; /* x får värdet 2.0 och n värdet 2 */
n = x = 2.4; /* x får värdet 2.4 och n värdet 2 */
```

Funktion

```
int
square (int n)
{
  return n * n;
}

int
sum_square (int a, int b)
{
  int a_sqr;
  int b_sqr;
  a_sqr = square (a);
  b_sqr = square (b);
  return a_sqr + b_sqr;
}
```

Explicit typkonvertering

- `(<typ>) <uttryck>`
  Detta kallas "type cast".
- Cast har högre prioritet än aritmetiska operatorer.:

```
int n = 1;
int m = 2;
float x;

x = (float) n / m;   /* x blir 0.5 */
x = (float) (n / m); /* x blir 0.0 */
```

If-sats

```
if (<villkor>)
  <sats>
```

```
if (<villkor>)
  <sats 1>
else
  <sats 2>
```

Exempel:

```
int
is_of_age (int age)
{
  if (age >= 18)
    return 1;
  else
    return 0;
}
```

While-slinga

```
while (<villkor>)
  <sats>
```

Exempel:

```
int sum = 0;
int i = 10;
while (i > 0) {
  sum = sum + i;
  i = i - 1;
}
```

In- och utmatning

Utskrift

```
printf(<sträng>);
printf(<formatsträng>, <variabel> [, <variabel2>, ...]);
```

Inmatning

```
scanf(<formatsträng>, &<variabel>, [, &<variabel2>, ...]);
```

Returnerar antalet inlästa värden

Formatsträng:

`%d`	heltal
`%f`	float
`%lf`	double

For-slinga

```
for (<init>; <villkor>; <uppdatering>)
  <sats>
```

Exempel:

```
int sum = 0;
int i;
for (i = 10; i > 0; i = i - 1)
  sum = sum + i;
```

In- och utmatning - exempel

```
printf("hej\n");
int pris = 790;
printf("Månadskort kostar %d kronor", pris);
float len = 168.5;
printf("Du är %f cm lång!",len);
antal = scanf("%d",&pris);
/* antal == 1 om värde inläst */
printf("priset är nu %d kronor.",pris);
antal = scanf("%d %f",&pris, &len);
printf("priset = %d, längden = %f", pris, len);
```

Do-slinga

```
do
  <sats>
while (<villkor>);
```

Exempel:

```
int sum = 0;
int i = 10;
do {
  sum = sum + i;
  i = i - 1;
} while (i > 0);
```

Användningsområde:

```
do {
  <ställ en fråga>
  <läs in svaret>
} while (<svaret var inte ok>);
```

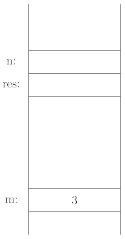
Ett komplett program

```
#include <stdio.h>

int main(){
  int vikt = 0;
  double langd = 0;
  printf("Ange din vikt och längd:");
  int antal = scanf ("%d %lf", &vikt, &langd);
  while (antal == 2) {
    double bmi = vikt / langd / langd;
    printf("Din bmi är %lf\n", bmi);
    printf("Ange en ny vikt och längd:");
    antal = scanf("%d %lf", &vikt, &langd);
  }
  printf("\nTack, hejdå!\n");
  return 0;
}
```

Avslutas med CTRL + D på tangentbordet vilket betyder "filslut".

Parameteröverföring (“call by value”)

```
#include <stdio.h>

int fact (int n) {
  int res;
  for (res = 1; n > 0; n = n - 1)
    res = res * n;
  return res;
}

int main () {
  int m = 3;
  printf (“%d", fact (m));
  return 0;
}
```



Deklaration av pekare

- I deklarationer innebär * att variabelnamnet representerar en pekare till den givna typen.

```
int v = 10; /* variabeln v får värdet 10 */

int *pek; /* här talar vi om för datorn att
             det som lagras i variabeln pek
             ska uppfattas som adress till
             ett minnesutrymme. */

pek = &v; /* här lagrar vi adressen till det
             värde som man kommer åt m.h.a
             variabeln v, i pekaren pek. */
```

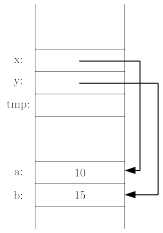
Exempel 2

Nu vill vi att variablerna a och b ska byta värde med varandra:

```
#include <stdio.h>
void swap (int x , int y) {
  int tmp = x;
  x = y;
  y = tmp;
}

int main () {
  int a=10;
  int b=15;
  swap (a, b); /* Fungerar inte! Varför? */
  printf (“a=%d och b=%d\n”, a, b);
  return 0;
}
```

Fungerande version av swap

```
#include <stdio.h>

void swap (int *x , int *y) {
  int tmp = *x;
  *x = *y;
  *y = tmp;
}

int main () {
  int a=10;
  int b=15;
  swap(&a, &b);
  printf(“a=%d och b=%d\n”, a, b);
  return 0;
}
```



Pekare

- `&<v>` ger adressen till minnesutrymmet där `v` lagras (ger en pekare till `v`). Exempel:

```
p = &x; /* p blir en pekare till x */
```

- `*<p>` ger det värde som finns i adressen `p` i minnet (följer pekaren). Exempel:

```
y = *p; /* y får det värde som p pekar på */
```

- Axiom: `*&x` är samma som `x`