

F08 - repetition av F5-F7

Programmeringsteknik med C och Matlab, 7,5 hp

Niclas Börlin
niclas.borlin@cs.umu.se

Datavetenskap, Umeå universitet

2023-10-10 Tis

- ▶ Fönstret för att anmäla sig till tentamen stänger i dag
- ▶ Instruktioner finns på <https://www.umu.se/student/mina-studier/tentamen/digital-salstentamen/>
- ▶ Om ni har fått beslut om pedagogiskt stöd, t.ex. i form av förlängd skrivtid
 - ▶ Skriv mail till niclas.borlin@cs.umu.se och studexp@cs.umu.se
 - ▶ **Bifoga** beslutet i en pdf-fil

Obligatorisk uppgift 2

- ▶ Inlämning
- ▶ Uppgiften
- ▶ Strukturerad problemlösning

Exempel (igen)

- ▶ Simulera 10000 kast med två tärningar och räkna hur ofta vi får respektive kombination av tärningar, spara resultatet i en tvådimensionell array och skriv ut resultatet när det är klart (se nedan)
- ▶ Algoritm:
 - ▶ Initiera arrayens element till 0
 - ▶ Upprepa 10000 gånger
 - ▶ Slå två tärningar
 - ▶ Öka värdet på arrayelementet som ges av de två tärningarnas värden med 1
 - ▶ Skriv ut resultatet

Endimensionella arrayer

- ▶ Har man relaterat data kan man lagra detta i arrayer istället för i enkla variabler
- ▶ Arrayerna deklareras och indexeras med hjälp av hakparenteser ([och]) direkt efter variabelnamnet
- ▶ Arrayerna har en fix längd n och indexeras från 0, dvs. 0, 1, ..., $n-1$

```
int x0;  
int x1;  
int x2;  
x0 = 2;  
x1 = 1;  
x2 = x0;
```

```
int x[3];  
  
x[0] = 2;  
x[1] = 1;  
x[2] = x[0];
```

Endimensionella vektorer

- ▶ Vektorer kan liksom enkla variabler *initieras* med värden vid deklarationen

```
int x0 = 2;  
int x1 = 1;  
int x2 = 4;
```

```
int x[3] = {2, 1, 4};
```

- ▶ Om listan med initieringsvärden är kortare än vektorn, initieras resterande element till 0

```
// set first element of x to 1, the rest to 0  
int x[100] = {1, 0};  
// zero all elements in y  
int y[100] = {0};
```

Funktioner i C

- ▶ En funktion i C är ett stycke kod som
 - ▶ har ett namn
 - ▶ tar ett specifikt antal parametrar (≥ 0), var och en av en specificerad typ
 - ▶ "gör något"
 - ▶ returnerar inget eller ett värde av en specifik typ
- ▶ Exempel:

```
int add(int n, int m)  
{  
    int result;  
    result = n + m;  
    return result;  
}
```

Funktionsdeklarationer

- ▶ En funktion ska deklareras innan den används
- ▶ Kallas också funktionsprototyp
- ▶ En funktionsprototyp berättar för kompilatorn
 - ▶ antalet argument
 - ▶ vilken typ argumenten har
 - ▶ vilken typ av värde (om något) som funktionen returnerar
- ▶ Exempel: Deklarationen

```
double sqrt(double);
```

berättar för kompilatorn att `sqrt` är en funktion som tar en `double` som argument och att den returnerar ett värde av typen `double`

Funktionsprototyper

- ▶ Identifierare som anges i parameterlistan i en funktionsprototyp påverkar inte deklarationen utan är enbart av dokumentationssyfte
- ▶ Bägge nedanstående deklarationer (prototyper) är korrekta
 - ▶ Den första är mer informativ

```
void print_time(int hour, int min);
```

```
void print_time(int, int);
```

Funktionsanrop

- ▶ Programmet börjar alltid med ett anrop till `main()`
- ▶ När en funktion anropas, ges kontroll till den funktionen
- ▶ Efter att funktionen är klar med sitt jobb, ges kontrollen tillbaka till den anropande funktionen
- ▶ Kompilatorn kontrollerar att typerna på argumenten är kompatibla
- ▶ Argumenten skickas enligt *call-by-value*
 - ▶ Detta betyder att varje argument evalueras och värdet av argumentet skickas till funktionen
- ▶ Värdena lagras i motsvarande lokala variabel
- ▶ Alltså, om en variabel skickas till en funktion så ändras inte värdet på variabeln i den anropande miljön

Parametrar

- ▶ Vi definierar de *formella* parametrarna till en funktion
- ▶ De parametrar som skickas till funktionen vid anrop kallas *aktuella* parametrar (argument)
- ▶ De aktuella parametrarna kan utgöras av uttryck
- ▶ Uttrycket evalueras i så fall innan anropet sker
- ▶ Vilken aktuell parameter som knyts till vilken formell parameter styrs enbart av ordningen i anropet och deklarationen

Pekare, lite repetition...

- ▶ Varje variabel i ett program har en unik *minnesadress*
- ▶ Med pekare kan man referera och manipulera minnesadresser
- ▶ Värdet i en *pekarvariabel* är minnesadressen till en variabel
- ▶ Om vi har en variabel `v` så anger `&v` minnesadressen till `v`
- ▶ Deklarationer av pekarvariabler görs med en `*` före variabelnamnet

```
int *ip;  
char *cp;  
double *dp;
```

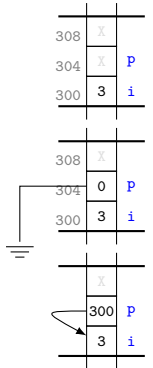
Pekare

- ▶ Bland de tillåtna adresserna att *referera* till finns alltid adressen 0
- ▶ Adressen 0 har i C ett speciellt namn, nämligen `NULL`
- ▶ Adressen `NULL` används som signalvärde hos pekare då inga variabler kan ha den adressen, d.v.s. `&v` kan aldrig returnera 0 (`NULL`)
- ▶ Några exempel på tilldelning av pekare kan vara:

```
int i;  
int *p = NULL;  
p = &i;
```

Pekare

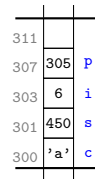
```
int i = 3;  
int *p;  
  
int i = 3;  
int *p = NULL;  
  
int i = 3;  
int *p = &a;
```



Pekare och minnet

- ▶ Pekaren håller alltså reda på i vilken byte ett värde "börjar"
- ▶ Exempel:

```
char c = 'a';  
short int s = 450;  
int i = 6;  
int *p = &i;
```



Pekare

- ▶ Man kan referera värdet på variabler med hjälp av pekare
- ▶ Om `p` är en pekare som pekar på `b`'s minnesadress, ger `*p` värdet av `b`
 - ▶ Man säger att pekaren *derefereras*
- ▶ Exempel:

```
int b = 3;  
int *p = &b; // Initialize p to the address of b  
printf(" b: %d\n p: %p\n*p: %d\n", b, p, *p);
```

Ger utskriften:

```
b: 3  
p: effffbac // printed in hexadecimal form  
*p: 3
```

Pekare

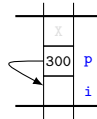
► Kodan

```
int a = 1;
int *p = &a;

printf(" a = %d *p = %d\n\n", a, *p);

*p = 2; // equivalent to a = 2
printf("a = %d *p = %d\n\n", a, *p);

a = 3; // equivalent to *p = 3
printf(" a = %d *p = %d\n", a, *p);
```



ger utskriften:

```
a = 1 *p = 1
a = 2 *p = 2
a = 3 *p = 3
```

Resultat från funktioner

► Via **return**

- Enkla funktioner med ett resultat, ex.
fabs(y);

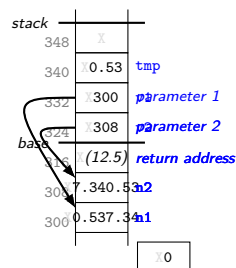
► Via parameterar

- Om vi vill ha ut flera resultat från en funktion
Ex: void swap(double *v1, double *v2)

- Ofta används returvärdet som en signal om allt gick som det skulle eller inte om resultat returneras via parametrar
Ex: scanf();

Returnera värden via parametrar

```
code/swap.c
1  #include <stdio.h>
2  void swap(double *v1, double *v2)
3  {
4      double tmp;
5      tmp = *v1;
6      *v1 = *v2;
7      *v2 = tmp;
8  }
9  int main(void)
10 {
11     double n1 = 0.53, n2 = 7.34;
12     swap(&n1, &n2);
13     printf("n1 = %4.2f, n2 = %4.2f\n", n1, n2);
14     return 0;
15 }
```



Ett exempel

► Skriv ett program som

- Skapar en 2-dimensionell array
- Nollställer arrayen
- Fyller arrayen med slumpstal mellan min och max
- Skriver ut arrayen
- Beräknar medelvärdet för alla tal i arrayen
- Summerar alla positiva och alla negativa tal, samt räknar antalet positiva och negativa tal
- Skriver ut resultatet