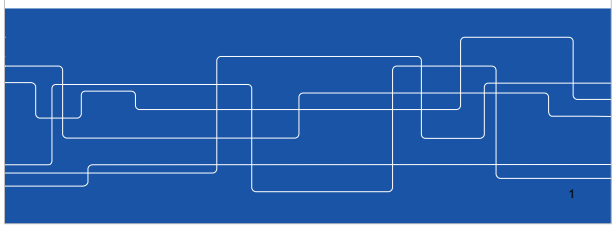
Föreläsning 14

Strängar, struct, länkad lista, filer



Standardfunktioner stränghantering

I stdio.h finns också funktionen sscanf som läser in värden till variabler från en sträng:

```
int sscanf (char *str, char *format, ...);
```
Returnera antal inlästa värden.

Exempel:

```
char *buf[] = "1 3.14";
int n, d;
double x;
n = sscanf(buf, "%d %lf", &d, &x);
/* d är nu 1 och x 3.14 */
```

Tvåkomplementsform

Litet bakgrund inför labb 2:

- Tvåkomplementsform är ett sätt att representera positiva och negativa tal i det binära talsystemet.
- Givet 8 bitar representeras talen -128 till -1 med de binära talen `10000000` till `11111111` medan 0 till 127 representeras av `00000000` till `01111111`.
- Givet n bitar kan alltså talen -2^{n-1} till $2^{n-1}-1$ representeras.
- För datatypen int är n = 32 och talomfånget blir -2147483648 till 2147483647.

Egen datastruktur (struct)

Ett sätt att beskriva sammansatta objekt
(liknar vagt klass i Python)

Deklaration av struct-typ:

```
struct <namn> {
  <fältdeklaration1>
  <fältdeklaration2>
  ...
};
```

Exempel:

```
struct C {
  int a;
  int b;
};
```

Definition av struct-variabel:

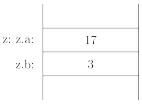
```
struct <namn> <variabel>;
```

Fältaccess (används som variabel):

```
<variabel>.<fältnamn>
```

Exempel:

```
struct C z;
z.a = 17;
z.b = 3;
```



Standardfunktioner stränghantering

I string.h finns ett antal standardfunktioner för stränghantering.
Här är några:

```
size_t strlen (char *s);
```
Returnera antal tecken i s ('\0' ej inräknat)

```
int strcmp (char *s1, char *s2);
```
Jämför s1 och s2 (lexikografiskt). Returnera negativt heltal om s1 < s2, positivt om s1 > s2 och noll om s1 och s2 är likvärdiga.

```
char *strcat (char *dest, char *src);
```
Förläng dest med src; returnera dest.

```
char *strcpy(char *dest, char *src);
```
Kopiera src till dest; returnera dest.

Egen datastruktur (struct)

```
struct Person {
  char name[20];
  int shoe_size;
}

struct Person mamma;

strcpy (mamma.name, "Emma");

mamma.shoe_size = 37;
```

Skapa nya typnamn med typedef

```
typedef <deklaration>
```

där `<deklaration>` ser ut som en variabeldefinition men där "definitionen" deklarerar en ny typ istället för variabel.

Exempel:

```
typedef int boolean;
typedef struct Person Person;
typedef char *String;
typedef int Intvector[10];
boolean found_key;
Person mamma;
String str = "hej hej";
Intvector vec;
```

Makron

```
#define N 20
#define SQR(n) n * n /* dåligt! */
```

Exempel:

```
y = SQR (2.5); /* y = 2.5 * 2.5 */
```

Problem:

```
k = SQR (i++); /* k = i++ * i++ */
x = SQR (1 + 2); /* x = 1 + 2 * 1 + 2 */
```

Ny definition:

```
#define SQR(n) ((n)*(n))
SQR(1+2) /* ((1+2)*(1+2)) */
```

Pekare till struct

```
struct Rational {
  int numerator;
  int denominator;
};
typedef struct Rational Rational;
int main () {
  Rational *ratpek =
    malloc (sizeof (Rational));
  ratpek->numerator = 10;
  ratpek->denominator = 5;
}
```

`pekare->fält` betyder alltså `(*pekare).fält`

Multipel inkludering

- Användning av makron för att undvika att deklarationer i en headerfil läses flera gånger då den inkluderas flera gånger

```
#ifndef RATIONAL_H
#define RATIONAL_H
...
gränssnitt för rationella tal (rational.h)
...
#endif /* RATIONAL_H */
```

Vektor av struct

```
struct Person pvek[10];
pvek[5].shoe_size = 42;
```

Initialisering:

```
Rational half = {1 , 2};
Rational ratvek[] ={{1, 3},{2, 5},{4, 3}};
```

Länkad lista

```
struct IntItem {
  int num;
  struct IntItem *tail;
};
typedef struct IntItem IntItem;
int sumList (IntItem *list) {
  int sum = 0;
  while (list != NULL) {
    sum += list->num;
    list = list->tail;
  }
  return sum;
}
```

Filhantering

```
FILE *f; /* filpekare */
f = fopen (“testdata.txt”, ”r”);
fscanf (f, “%d”, &n);
fclose(f);
```

Då programmet startar öppnas automatiskt filer för inmatning och utmatning. Dessa representeras av 3 fördefinierade filpekare:

`stdin` fil som inmatning tas ifrån, av tex `scanf`

`stdout` fil som utmatning går till, från tex `printf`

`stderr` fil som felmeddelanden går till

Dessa är av typen `FILE *` och deklareras i stdio.h.

Läsning / skrivning

```
char * fgets (char *s, int n, FILE *fil);
```

läser en rad (dock maximalt n - 1 tecken) och lägger tecknen i det minne som s pekar på. Returvärdet är s om allt gick bra, NULL om vid fel (tex att filslut påträffats innan något tecken lästs).

Exempel:

```
while (fgets (line, 80, f) != NULL) {
  if (isdigit (line[0])) {
    sscanf (line, “%d”, &n);
    sum += n;
  }
}
```

Öppna och stänga fil

En fil ska öppnas med `fopen` och stängs med `fclose`:

```
FILE *fopen (char *name, char *access);
int fclose (FILE *fil);
```

Accessmetoder (anges som sträng i andra argumentet)

`r` läsning

`w` skrivning

`a` tillägg i slutet på filen

`r+` läsning och skrivning (fil ska existera)

`w+` läsning och skrivning (filen töms)

`a+` läsning och skrivning (filpekare pekar på slutet)

Man lägger till ett `b` om filen ska hanteras binärt.

Läsning / skrivning

```
int fgetc (FILE *fil);
```

läser ett tecken från `fil` och returnerar det

```
int fputc (int tecken, FILE *fil);
```

skriver ett tecken på `fil`

```
int fprintf (FILE *fil, char *format, ...);
```

som printf men skriver på filen `fil`

```
int fscanf(FILE *fil, char *format,...);
```

som scanf men läser från filen `fil`

`getchar ()` samma som `fgetc (stdin)`

`putchar (c)` samma som `fputc (stdout, c)`