

```
/* funktionsdefinition */  
resultattyp funktionsnamn (typ1 param1, typ2 param2, etc.)  
{  
    lokala deklARATIONER och satser  
}
```

```
double upphojt_till(double x, int n)  
{  
    double res = 1;  
    int i = 1;  
    if (n >= 0)  
        for ( ; i <= n; i++)  
            res = res * x;  
    else  
        for ( ; i <= -n; i++)  
            res = res / x;  
    return res;  
}
```

```
double ranta_pa_ranta(double b, double r, int n)  
{  
    return b * upphojt_till(1+0.01*r, n);  
}
```

```
int bokstav(char c)    /* är c en bokstav ? */
{
    return ('a' <= c && c <= 'z') || ('A' <= c && c <= 'Z');
}
```

C99:

```
_Bool bokstav(char c)    /* är c en bokstav ? */
{
    return ('a' <= c && c <= 'z') || ('A' <= c && c <= 'Z');
}
```

```
int las_teck(void) /* läser första icke blanka */  
{  
    int c;  
    while ((c = getchar()) == ' ' || c == '\t' || c == '\n')  
        ;  
    return c;  
}
```

```
void blanka_rader(int n)  
{  
    for (int i=1; i<=n; i++)  
        printf("\n");  
}
```

```
/* funktionsdeklaration */  
resultattyp funktionsnamn (typ1 [param1], typ2 [param2], etc.);
```

```
double upphojt_till(double x, int n);
```

eller:

```
double upphojt_till(double, int);
```

```
double ranta_pa_ranta(double b, double r, int n);
```

eller:

```
double ranta_pa_ranta(double, double, int);
```

```
int las_teck(void);    // OBS! inte int las_teck();
```

```
void blanka_rader(int n);
```

eller:

```
void blanka_rader(int);
```

Rekursion

```
int nfak(int n) /* beräknar n! */
{
    if (n<=0)
        return 1;
    else
        return n * nfak(n - 1);
}
```

```
void skrivbak(char text[], int i)
/* skriver ut en text baklänges */
{
    if (text[i] != '\0') {
        skrivbak(text, i + 1);
        putchar(text[i]);
    }
}
```

```
skrivbak(s, 0);
```

Äldre syntax, definitioner:

```
double upphojt_till(x, n)
double x;
int n;
{
    ...
}
```

```
double ranta_pa_ranta(double b, double r, int n)
double b;
double r;
int n;
{
    ...
}
```

Äldre syntax, deklARATIONER:

```
double upphojt_till();
double ranta_pa_ranta();
```

```
inline int max( int i1, int i2 )  
{  
    return i1 < i2 ? i2 : i1;  
}
```

```
volatile int tangentbords_reg;  
extern volatile const int realtidsklocka;
```

Lagringsklasser

auto **register** **extern** **static**

```
extern double f(int);  
static int m = 1;  
register int i, j;
```

Standardantaganden:

För en funktion antas att lagringsklassen är **extern**.

För en formell parameter till en funktion antas att lagringsklassen är **auto**.

För en variabel som deklarerats inne i ett block antas att lagringsklassen är **auto**.

För en variabel som deklarerats på yttersta nivå antas att lagringsklassen är **extern**.


```
/* externa variabeldefinitioner */  
int antal = 5;  
const int max_antal = 100;  
const double pi = 3.1415926536;
```

```
/* externa variabeldeklarationer */  
extern int antal;  
extern const double pi;  
extern int n;  
extern const int max_antal;
```



```
// Filen strutil.c
#include <string.h>
#include <stdio.h>
#include "strutil.h"

void skip_line() {
    int c;
    while ((c=getchar()) != '\n' && c != EOF)
        ;
}

char *get_line(char *s, int n) {
    char *r = fgets(s, n, stdin);
    if (r)
        remove_newline(s);
    return r;
}

char *remove_newline(char *s) {
    char *pos = strchr(s, '\n');
    if (pos)
        *pos = '\0';
    return s;
}
```

