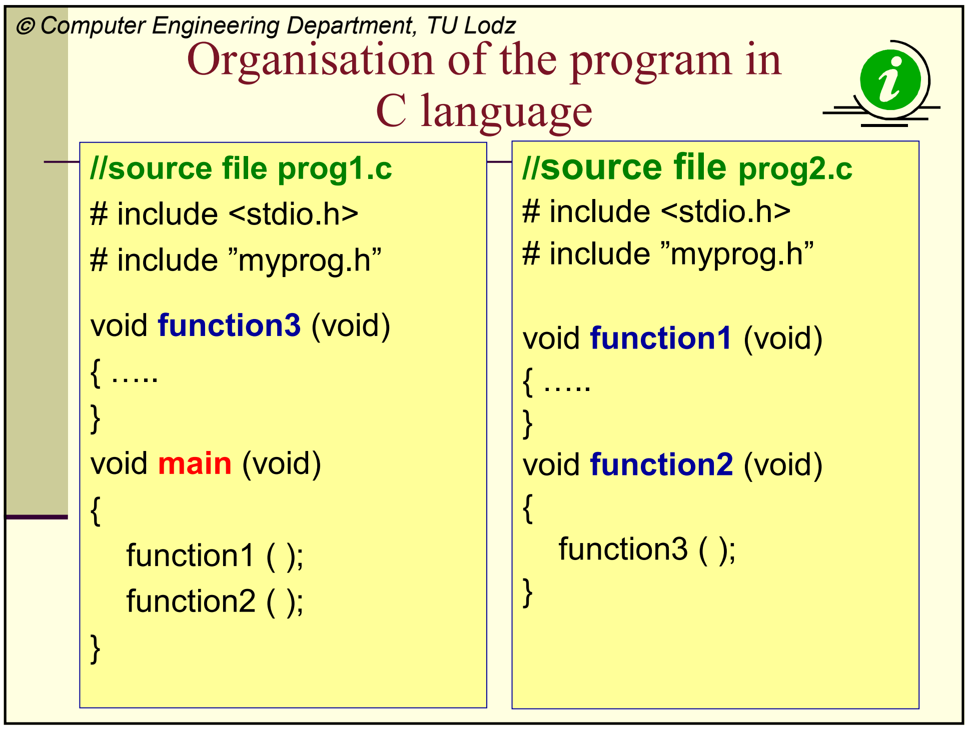
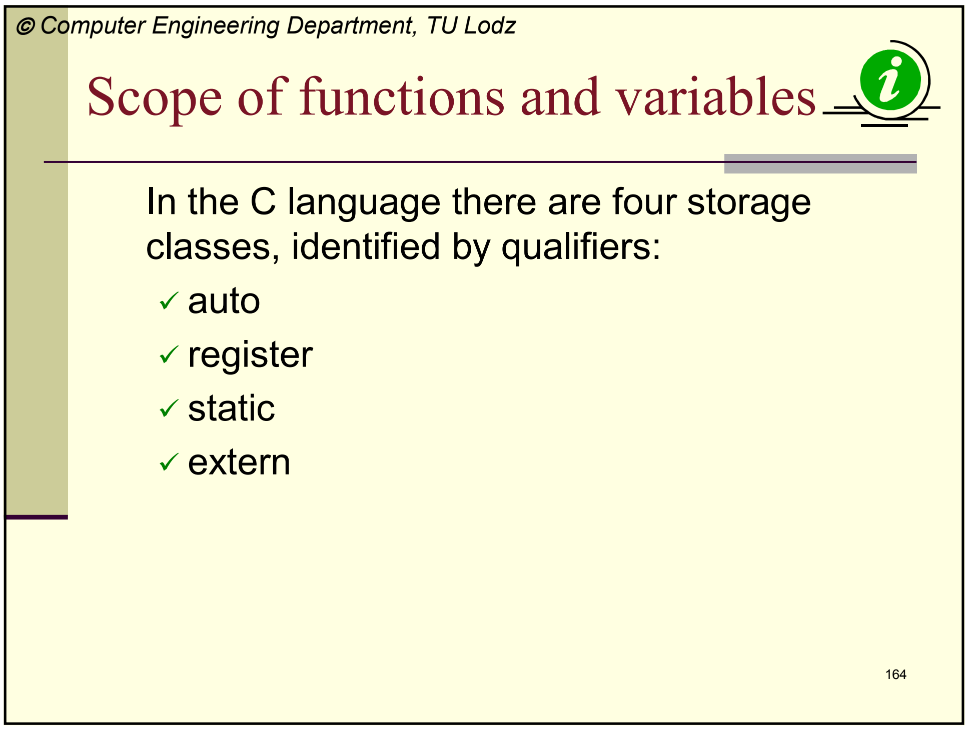


© Computer Engineering Department, TU Lodz

## Organisation of the program in C language

```
//source file prog1.c
# include <stdio.h>
# include ”myprog.h”

void function3 (void)
{ …..
}
void main (void)
{
    function1 ( );
    function2 ( );
}
```

```
//source file prog2.c
# include <stdio.h>
# include ”myprog.h”

void function1 (void)
{ …..
}
void function2 (void)
{
    function3 ( );
}
```



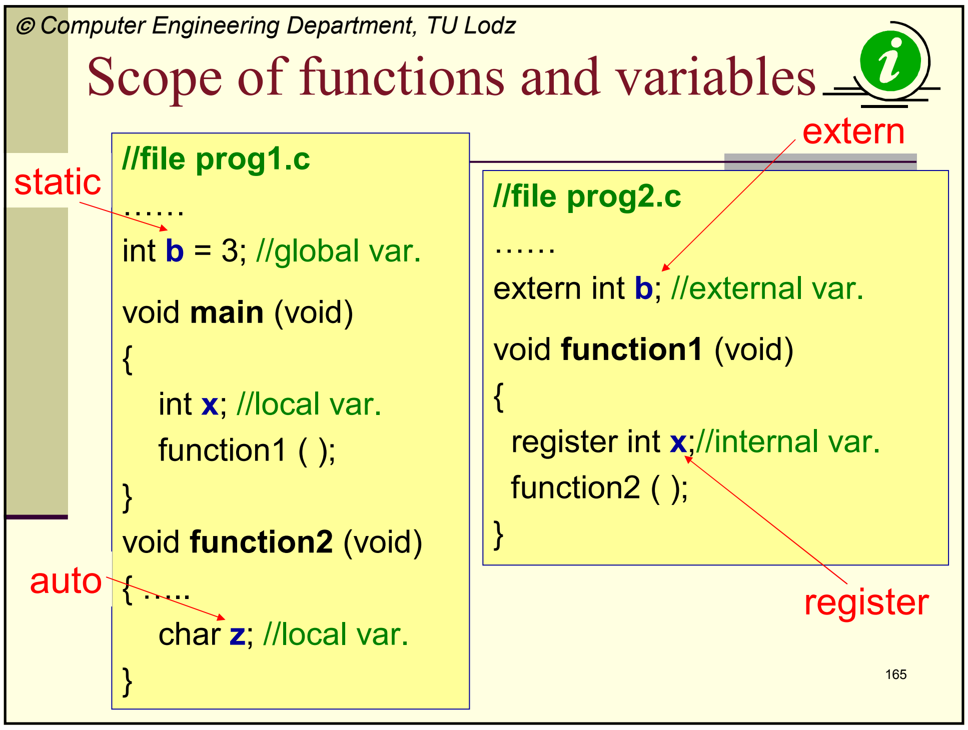
© Computer Engineering Department, TU Lodz

## Scope of functions and variables

In the C language there are four storage classes, identified by qualifiers:

- ✓ auto
- ✓ register
- ✓ static
- ✓ extern

164



© Computer Engineering Department, TU Lodz

## Scope of functions and variables

static

auto

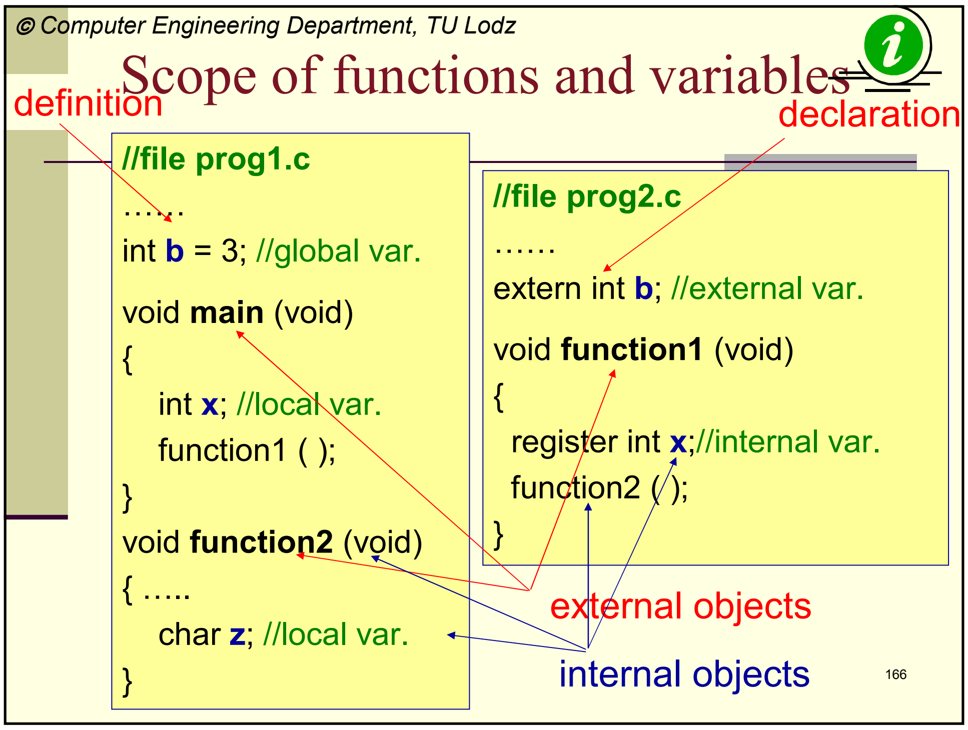
```
//file prog1.c
……
int b = 3; //global var.
void main (void)
{
    int x; //local var.
    function1 ( );
}
void function2 (void)
{ …..
    char z; //local var.
}
```

extern

```
//file prog2.c
……
extern int b; //external var.
void function1 (void)
{
  register int x;//internal var.
  function2 ( );
}
```

register

165



© Computer Engineering Department, TU Lodz

## Scope of functions and variables

definition

```
//file prog1.c
……
int b = 3; //global var.
void main (void)
{
    int x; //local var.
    function1 ( );
}
void function2 (void)
{ …..
    char z; //local var.
}
```

declaration

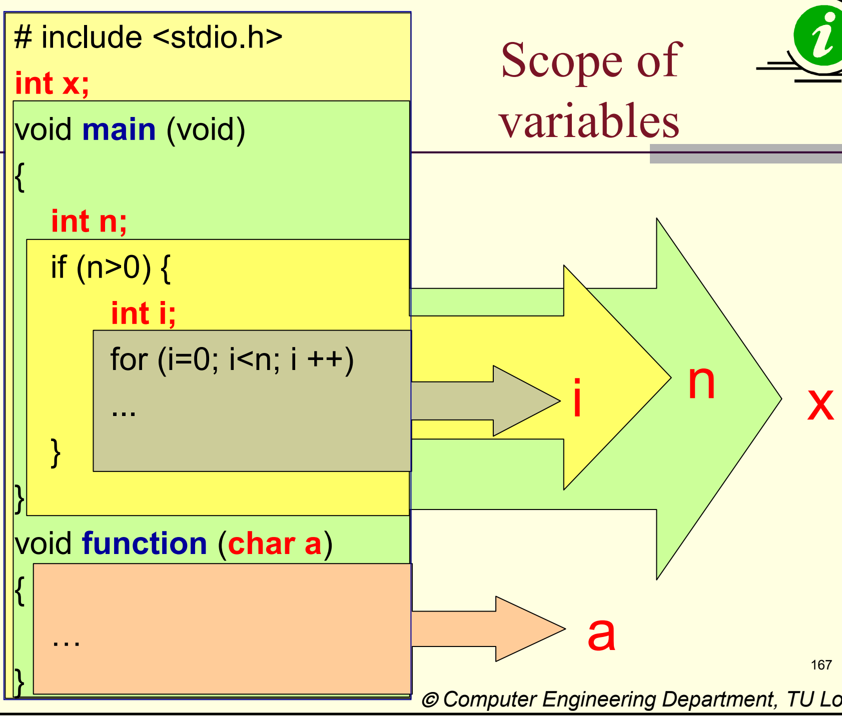
```
//file prog2.c
……
extern int b; //external var.
void function1 (void)
{
  register int x;//internal var.
  function2 ( );
}
```

external objects

internal objects

166

## Scope of variables



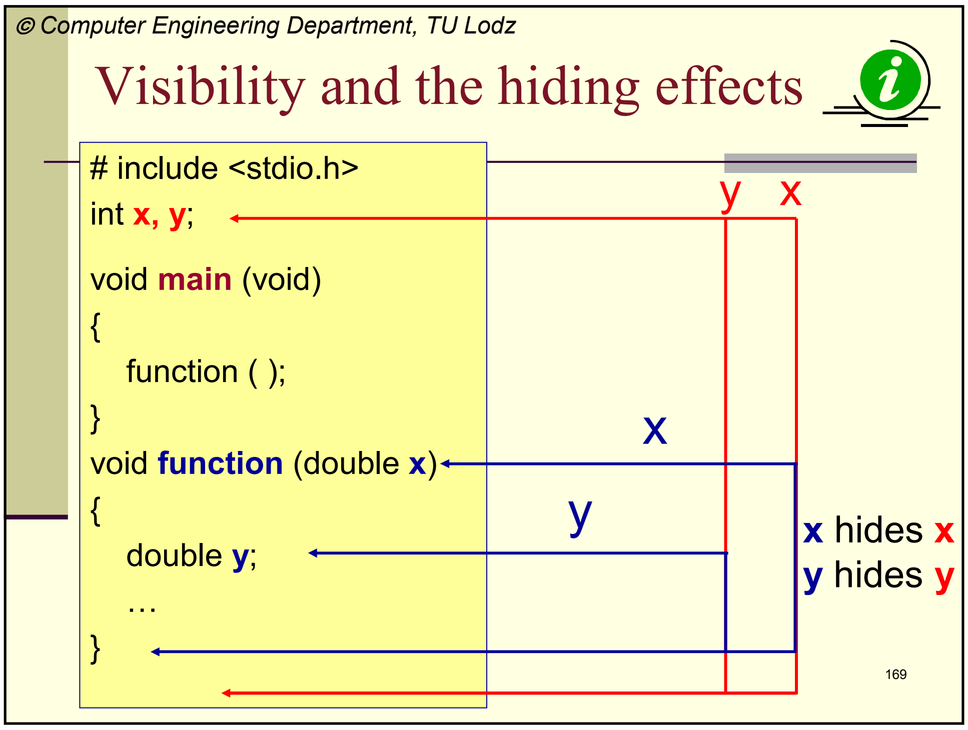
```
# include <stdio.h>
int x;
void main (void)
{
   int n;
   if (n>0) {
         int i;
         for (i=0; i<n; i ++)
         ...
   }
}
void function (char a)
{
   …
}
```

## Declarations and definitions

- **Declaration** defines the name of the object, its type and class, but doesn't allocate in the memory.
- **Definition** is a declaration, which additionally allocates the object in the place of the memory.

  It is possible to put the function declarations (prototypes) and external variable declarations in **the header files**.

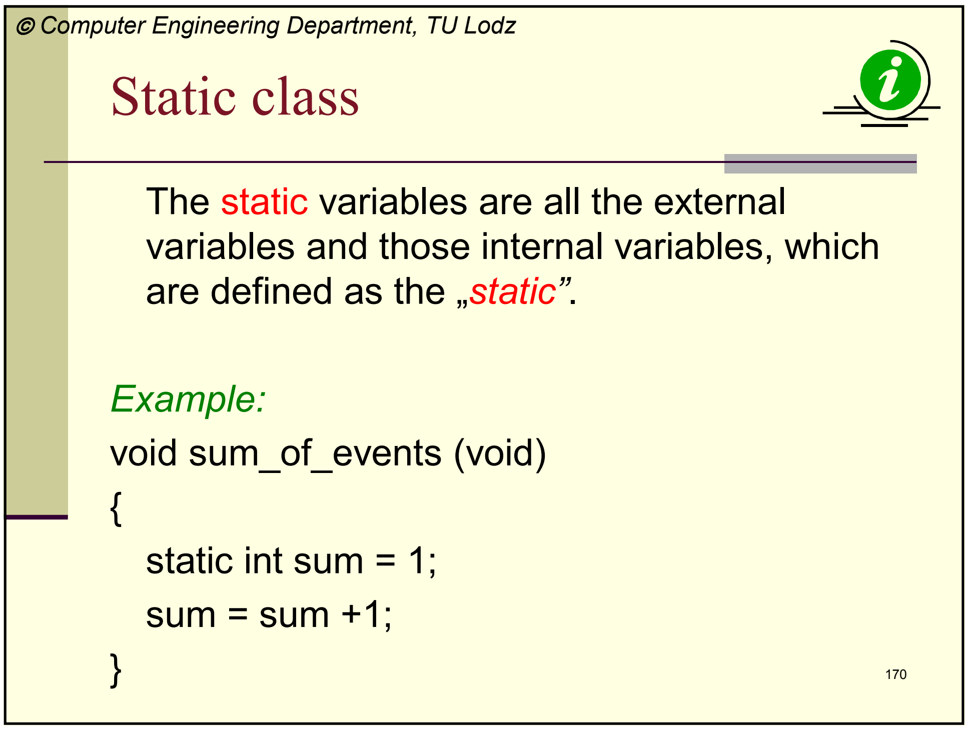
**You must not put the variable definitions in the header files !**


© Computer Engineering Department, TU Lodz

## Visibility and the hiding effects

```
# include <stdio.h>
int x, y;

void main (void)
{
    function ( );
}
void function (double x)
{
    double y;
    …
}
```

169


© Computer Engineering Department, TU Lodz

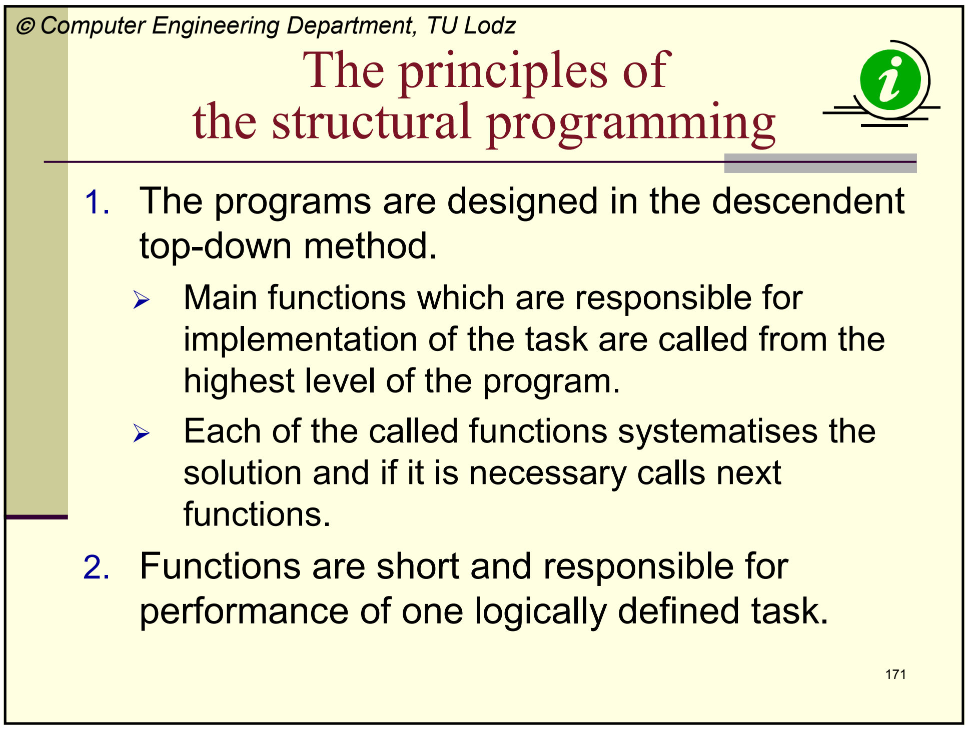
## Static class

The static variables are all the external variables and those internal variables, which are defined as the „*static*”.

*Example:*

```
void sum_of_events (void)
{
   static int sum = 1;
   sum = sum +1;
}
```

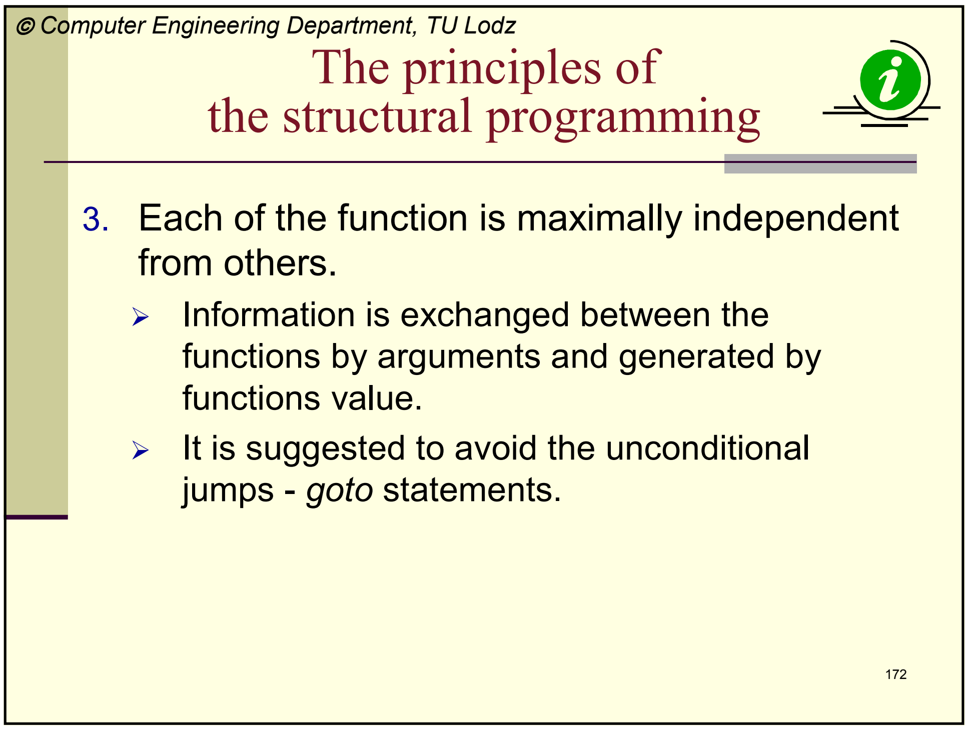
170



## The principles of the structural programming

1. The programs are designed in the descendent top-down method.
   - Main functions which are responsible for implementation of the task are called from the highest level of the program.
   - Each of the called functions systematises the solution and if it is necessary calls next functions.
2. Functions are short and responsible for performance of one logically defined task.

171



## The principles of the structural programming

3. Each of the function is maximally independent from others.
   - Information is exchanged between the functions by arguments and generated by functions value.
   - It is suggested to avoid the unconditional jumps - *goto* statements.

172

## *Example 25*

The following sequence $x_1, x_2, \dots, x_n$ ($n < 201$) is with the integer elements. Sort the sequence in a non-decreasing order.

The $\{ x_n \}$ sequence is ordered non-decreasingly if for each $i < n$ occurs: $x_i \leq x_{i+1}$.

## Bubble Sort

- While comparing two succeeding elements of the sequence subsequently for $i=1,2,\dots,n-1$ and swapping them if the inequality $x_i>x_{i+1}$ is true, will cause the moving of the largest element in the final position after one sequence pass.
- The similar swapping series can be repeated in the sequence decreased by one last element. As a consequence after the second pass the last two elements of the sequence will be in the right places. While continuing such a swapping series for smaller and smaller subsequences will order all the elements in the $\{x_n\}$ sequence.

## Bubble Sort cont.

- After the *k* passes the final *k* elements must be in the proper positions in the sequence (in bold). The previous positions were not necessarily ordered.
- Such a method is recommended for checking if the sequence is ordered and for putting the sequence in order, if swapping of minor numbers is expected, namely if the order was corrupted only in a few positions.

## Bubble sort, example

| x[1] | x[2] | x[3] | x[4] | x[5] | x[6] | x[7] | x[8] | comments |
|---|---|---|---|---|---|---|---|---|
| 6 | 2 | 7 | 1 | 8 | 3 | 9 | 5 | original sequence |
| 2 | 6 | | | | | | | swap x[1] and x[2] |
| | | 1 | 7 | | | | | swap x[3] and x[4] |
| | | | | 3 | 8 | | | swap x[5] and x[6] |
| | | | | | | 5 | 9 | swap x[7] and x[8] |
| 2 | 6 | 1 | 7 | 3 | 8 | 5 | **9** | after one pass |

| x[1] | x[2] | x[3] | x[4] | x[5] | x[6] | x[7] | x[8] | comments |
|---|---|---|---|---|---|---|---|---|
| 2 | 6 | 1 | 7 | 3 | 8 | 5 | **9** | after one pass |
|  | 1 | 6 |  |  |  |  |  | swap x[2] and x[3] |
|  |  |  | 3 | 7 |  |  |  | swap x[4] and x[5] |
|  |  |  |  |  | 5 | 8 |  | swap x[6] and x[7] |
| 2 | 1 | 6 | 3 | 7 | 5 | **8** | **9** | after two passes |

| x[1] | x[2] | x[3] | x[4] | x[5] | x[6] | x[7] | x[8] | comments |
|---|---|---|---|---|---|---|---|---|
| 2 | 1 | 6 | 3 | 7 | 5 | **8** | **9** | after two passes |
| 1 | 2 |  |  |  |  |  |  | swap x[1] and x[2] |
|  |  | 3 | 6 |  |  |  |  | swap x[3] and x[4] |
|  |  |  |  | 5 | 7 |  |  | swap x[5] and x[6] |
| 1 | 2 | 3 | 6 | 5 | **7** | **8** | **9** | after 3 passes |
|  |  |  | 5 | 6 |  |  |  | swap x[4] and x[5] |
| 1 | 2 | 3 | 5 | **6** | **7** | **8** | **9** | after 4 passes |

## Program 1

```
// header file thisprog.h
/* program illustrating  organisation of the
   program in C language */

void sort_b (int, int[]); //function prototype
void printing (int, int[]);
```

```
// source file sorting.c
/* program illustrating  organisation of the
   program in C language */

# include <stdio.h>
# include <stdlib.h>
# include "thisprog.h"


void sort_b (int n, int a[ ])
{
   int p,i,aa;
   do {
      p=0;
      for (i=0; i<n-1; i++) {
           if (a[i]>a[i+1])
           {
                aa=a[i];
                a[i]=a[i+1];
                a[i+1]= aa;
                p=1; //swap
           }
      }
   } while (p!=0);
}
```

```
// source file program.c
/* program illustrating  organisation of the
   program in C language */

# include <stdio.h>
# include <stdlib.h>
# include "thisprog.h"

int main (void)
{
   int n,j;
   int a[200];
   char c;
```

```
system("cls"); //stdlib.h – clearing the screen
fflush(stdin); //stdio.h – clearing the buffer
printf("Program for sorting the integer elements
of the sequence\n\n");
printf("Enter the number of the sequence
elements n<201 ");
scanf("%d", &n);
printf("\nEnter the consecutive sequence
elements \n\n");
for (j=0; j<n; j++) {
        printf("a[%d] = ", j+1);
        scanf("%d", &a[j]);
}
```

```
  system ("cls");
  printf ("The given sequence:\n\n");
  printing (n,a);
  printf ("\n\n");
  sort_b (n,a);
  printf ("The sequence sorted non-
  decreasingly:\n\n");
  printing (n,a);
  printf ("\n\n");
  c = getchar();

return 0;
}
```

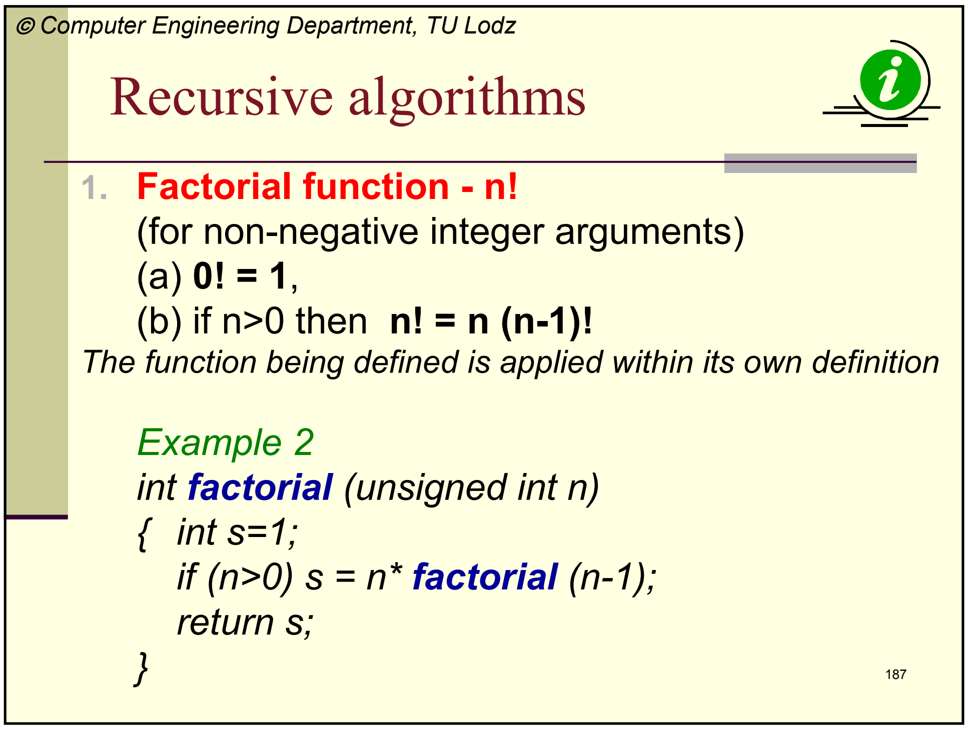
```
void printing (int k, int b[])
{
   int i;

   for (i=0; i<k; i++) {
        printf("  %6d", b[i]);
        if ((i+1)%10==0) printf("\n");
   }
}
```

## Iterative algorithms

1. **Factorial function - n!**
   (for non-negative integer arguments)
   (a) **0! = 1**,
   (b) **n! = 1·2·3·…·(n-1)·n,** for n>0

*Example 1*

```
int i,s,n;
i=0; s=1;
while (i<n) {
     i=i+1;    s = s*i;
}
```



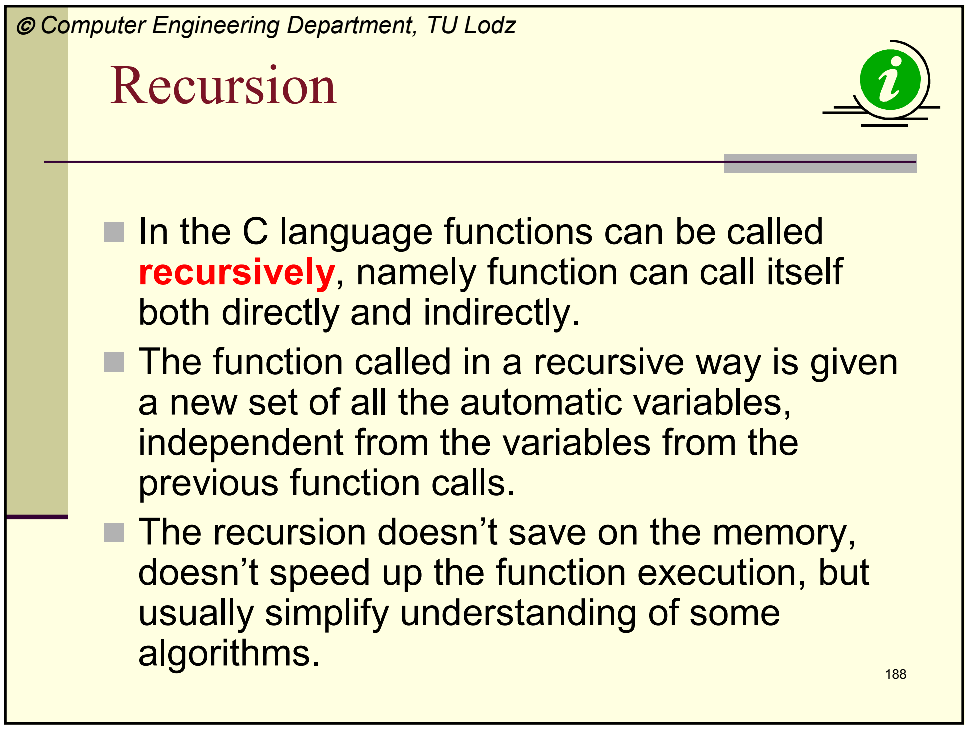
## Recursive algorithms

1. **Factorial function - n!**
   (for non-negative integer arguments)
   (a) **0! = 1**,
   (b) if n>0 then **n! = n (n-1)!**

*The function being defined is applied within its own definition*

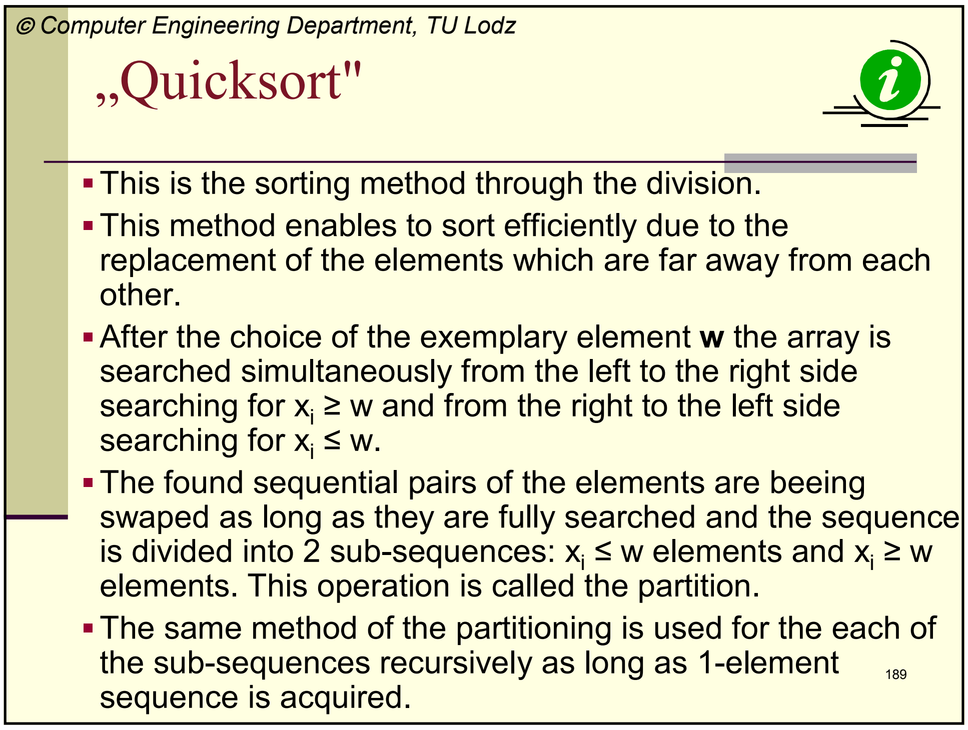
*Example 2*

```
int factorial (unsigned int n)
{   int s=1;
    if (n>0) s = n* factorial (n-1);
    return s;
}
```



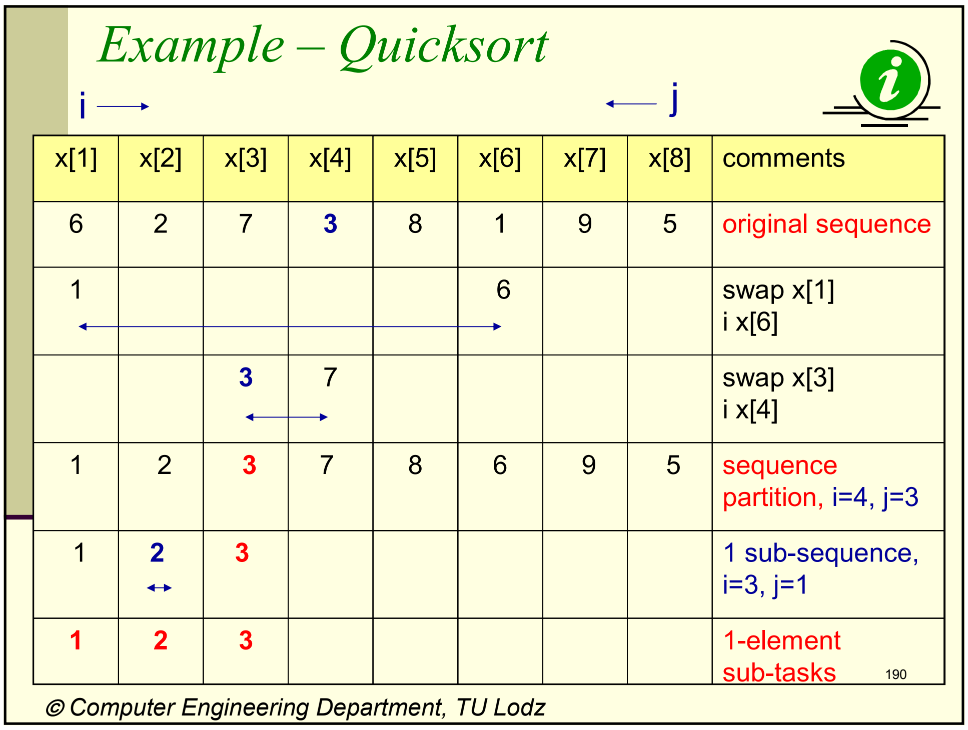
## Recursion

- In the C language functions can be called **recursively**, namely function can call itself both directly and indirectly.
- The function called in a recursive way is given a new set of all the automatic variables, independent from the variables from the previous function calls.
- The recursion doesn’t save on the memory, doesn’t speed up the function execution, but usually simplify understanding of some algorithms.



## „Quicksort"

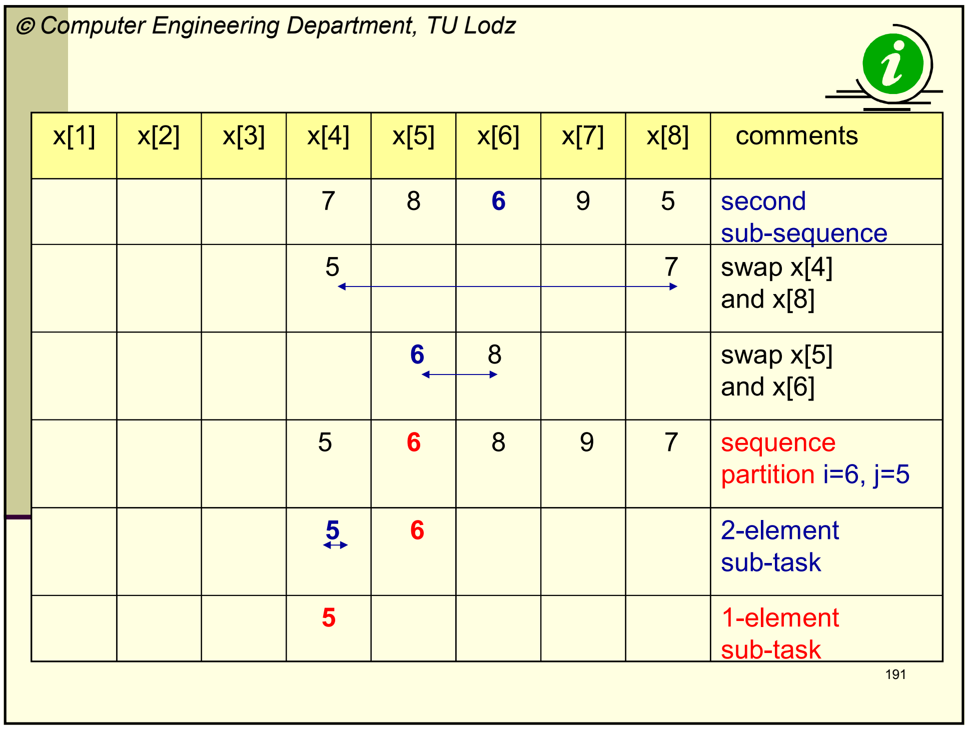
- This is the sorting method through the division.
- This method enables to sort efficiently due to the replacement of the elements which are far away from each other.
- After the choice of the exemplary element **w** the array is searched simultaneously from the left to the right side searching for $x_i \geq w$ and from the right to the left side searching for $x_i \leq w$.
- The found sequential pairs of the elements are beeing swaped as long as they are fully searched and the sequence is divided into 2 sub-sequences: $x_i \leq w$ elements and $x_i \geq w$ elements. This operation is called the partition.
- The same method of the partitioning is used for the each of the sub-sequences recursively as long as 1-element sequence is acquired.



## Example – Quicksort

i ⟶ ⟵ j

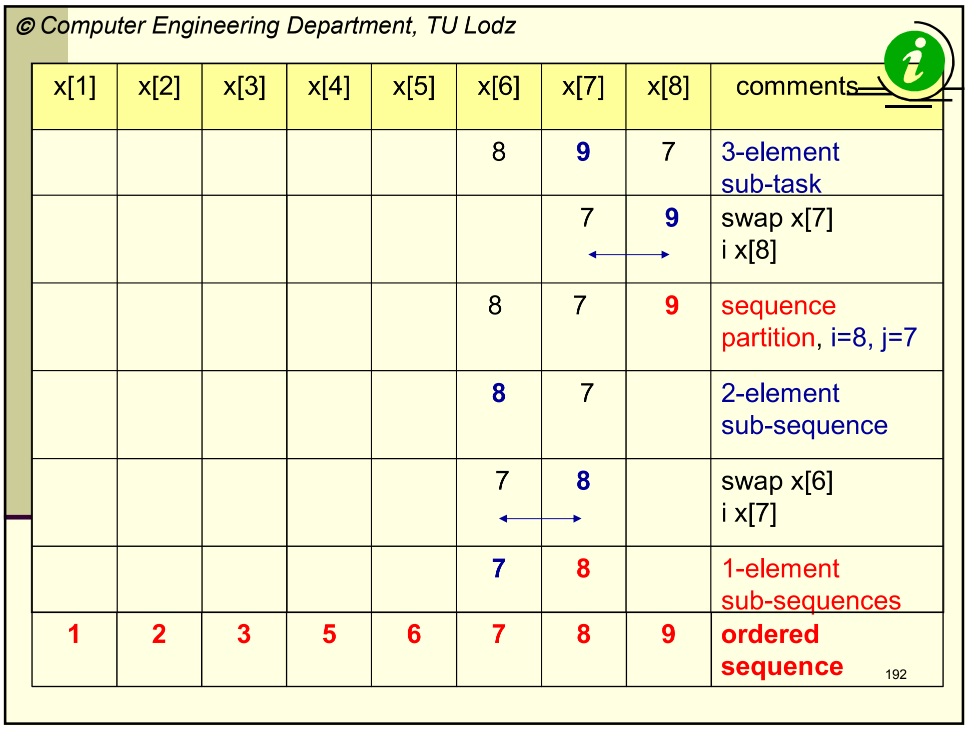
| x[1] | x[2] | x[3] | x[4] | x[5] | x[6] | x[7] | x[8] | comments |
|---|---|---|---|---|---|---|---|---|
| 6 | 2 | 7 | **3** | 8 | 1 | 9 | 5 | original sequence |
| 1 | | | | | 6 | | | swap x[1] i x[6] |
| | | **3** | 7 | | | | | swap x[3] i x[4] |
| 1 | 2 | **3** | 7 | 8 | 6 | 9 | 5 | sequence partition, i=4, j=3 |
| 1 | **2** | **3** | | | | | | 1 sub-sequence, i=3, j=1 |
| **1** | **2** | **3** | | | | | | 1-element sub-tasks |



ⓒ *Computer Engineering Department, TU Lodz*

| x[1] | x[2] | x[3] | x[4] | x[5] | x[6] | x[7] | x[8] | comments |
|---|---|---|---|---|---|---|---|---|
| | | | 7 | 8 | **6** | 9 | 5 | second sub-sequence |
| | | | 5 | | | | 7 | swap x[4] and x[8] |
| | | | | **6** | 8 | | | swap x[5] and x[6] |
| | | | 5 | **6** | 8 | 9 | 7 | sequence partition i=6, j=5 |
| | | | **5** | **6** | | | | 2-element sub-task |
| | | | **5** | | | | | 1-element sub-task |

191



ⓒ *Computer Engineering Department, TU Lodz*

| x[1] | x[2] | x[3] | x[4] | x[5] | x[6] | x[7] | x[8] | comments |
|---|---|---|---|---|---|---|---|---|
| | | | | | 8 | **9** | 7 | 3-element sub-task |
| | | | | | | 7 | **9** | swap x[7] i x[8] |
| | | | | | 8 | 7 | **9** | sequence partition, i=8, j=7 |
| | | | | | **8** | 7 | | 2-element sub-sequence |
| | | | | | 7 | **8** | | swap x[6] i x[7] |
| | | | | | **7** | **8** | | 1-element sub-sequences |
| **1** | **2** | **3** | **5** | **6** | **7** | **8** | **9** | **ordered sequence** |

192

## Program 2

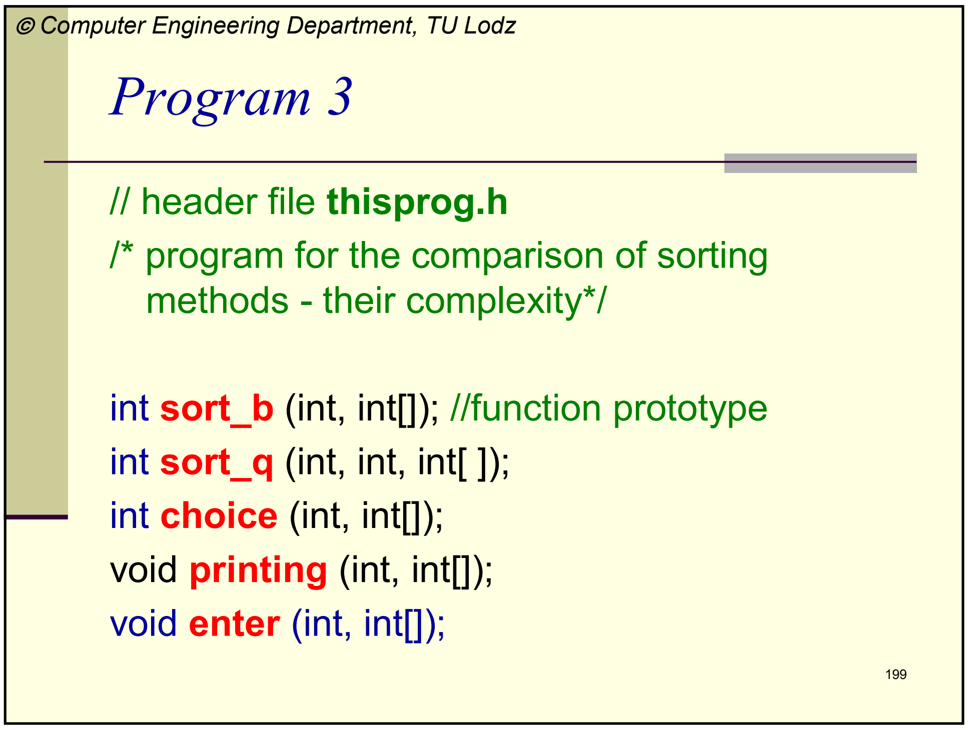
```
// header file thisprog.h
/* program for sorting the sequence */

void sort_b (int, int[]); //function prototype
void sort_q (int, int, int[ ]);
void choice(int, int[]);
void printing (int, int[]);
```

```
// source file sorting.c
# include <stdio.h>
# include <stdlib.h>
# include "thisprog.h"
void sort_b (int n, int a[ ]) //bubble sorting
{
   …
}
void sort_q (int l, int p, int a[ ]) //quick sorting
{
   int i,j,aa,w;
   i=l; j=p;
   w=a[(l+p)/2];
   do {
         while (a[i]<w)  i++;
         while (w<a[j])  j--;
         if (i<=j)
                 {
                    aa=a[i];           //swaps a[i] and
   a[j]
                    a[i]=a[j]; a[j]= aa;
                    i++; j--;
                 }
   } while (i<=j);
   if (l < j) sort_q (l, j, a);
   if (i < p) sort_q (i, p, a);
}
```

```
void choice (int n, int a[])
/* choice of the sorting method */
{
   int p;
   char c;

   do {
         fflush(stdin);  //stdio.h – clearing the buffer
         p=0;
         printf("choose the sorting method:
                                b - bubble, q - quick\n");
         c = getchar();
```

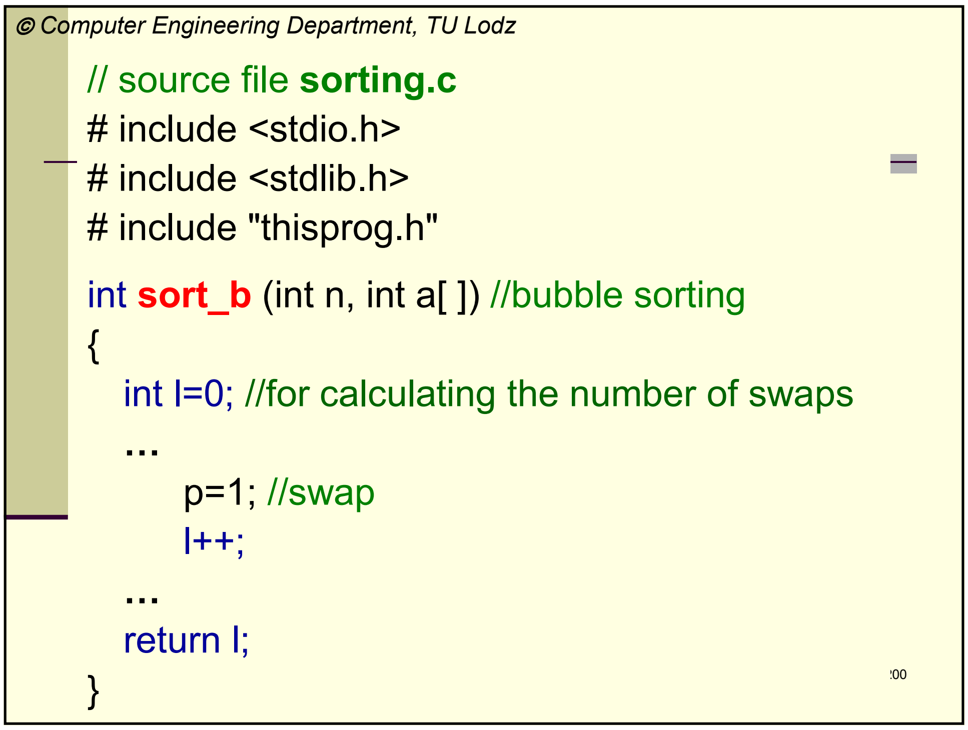
```
        switch (c)
        {
            case 'b': sort_b (n,a);
                      break;
            case 'q': sort_q (0,n-1,a);
                      break;
            default: printf ("Error!   "); p=1;
                      break;
        }
    } while (p);
}
```

```
// source file program.c

# include <stdio.h>
# include <stdlib.h>
# include "thisprog.h"

int main (void)
{
   ...
   /* sort_b (n,a); */  choice (n,a);
   ...
}
```



## Program 3

```
// header file thisprog.h
/* program for the comparison of sorting
   methods - their complexity*/

int sort_b (int, int[]); //function prototype
int sort_q (int, int, int[ ]);
int choice (int, int[]);
void printing (int, int[]);
void enter (int, int[]);
```



```
// source file sorting.c
# include <stdio.h>
# include <stdlib.h>
# include "thisprog.h"
int sort_b (int n, int a[ ]) //bubble sorting
{
   int l=0; //for calculating the number of swaps
   …
        p=1; //swap
        l++;
   …
   return l;
}
```

```
int sort_q (int l, int p, int a[ ]) //quicksort
{
    static int lz=0; // for calculating the number of
    swaps
    int i,j,aa;
    i=l; j=p;
    register int w;
    w=a[(l+p)/2];
```

```
    do {
        while (a[i]<w)  i++;
        while (w<a[j])  j--;
        if (i<=j)
                {
                  aa=a[i];             //swap a[i] and a[j]
                  a[i]=a[j]; a[j]= aa;
                  i++; j--; lz++;
                }
    } while (i<=j);
    if (l < j) sort_q (l, j, a);
    if (i < p) sort_q (i, p, a);
    return lz;
}
```

```
int choice (int n, int a[])
/* choice of the sorting method */
{
   int p, l;
   char c;

   do {
        fflush(stdin);  //stdio.h – clearing the buffer
        p=0;
        printf("choose the sorting method:
                          b - bubble, q - quick\n");
        c = getchar();
```

```
        switch (c)
        {
                case 'b': l = sort_b (n,a);
                          break;
                case 'q': l = sort_q (0,n-1,a);
                          break;
                default: printf ("Error!   "); p=1;
                          break;
        }
   } while (p);
   return l; //number of swaps
}
```

```
// source file program.c
# include <stdio.h>
# include <stdlib.h>
# include "thisprog.h"

int main (void)
{
   int n, lw;
   int a[200];
   char c;
   …
   enter (n,a); // successive elements of the
   sequence
```

```
   …
   printing(n,a);
   …
   lw = choice (n,a);   /* sort_b (n,a);  choice
   (n,a);*/
   …
   printing(n,a);
   …
   printf("Number of swaps = %d\n",lw);
   printf("\n");
   c = getchar();
return 0;
}
```

```
void enter (int n, int a[])
{
   char c;
   int j;
   unsigned int seed;
   printf("Choose the method of inserting the sequence
   of numbers\n");
   printf("\tk – from the keyboard,\n\tg – from the
   generator of random numbers\n\n");
   fflush(stdin); //stdio.h – clearing the buffer
   c=getchar();
```

```
while (c!='k'&& c!='g') {
        printf("Error!  Enter again\n");
        fflush(stdin);
        c=getchar();
   }
   if (c=='k') {  //part of the old program:
        printf("Enter the  consecutive elements of
   the sequence \n\n");
        for (j=0; j<n; j++) {
                printf("a[%d] = ", j+1);
                scanf("%d", &a[j]);
        } //end of the part
   }
```

```
    else {
        printf("\nEnter seed ");
        scanf("%d", &seed);
        printf("\n\n");
        srand(seed); //stdlib.h
        for (j=0; j<n; j++) a[j]=rand(); //stdlib.h
    }
}

void printing (int k, int b[])
{  …  }
```

## *Example 26*

*Write the program finding the greatest common divisor of two natural numbers.*
*Use Euclide's recursive algorithm.*

```
/* program finding the greatest common
divisor of two natural numbers */
# include <stdio.h>
int GCD (int, int);

int main (void)
{
    int w,x,y;
    printf ("Program finding the greatest common divisor of
    two natural numbers \n\n");
    printf ("Enter two natural numbers");
    scanf ("%d %d", &x, &y);
    w = GCD (x,y);
    printf ("Greatest common divisor of numbers %d and
    %d is %d\n", x,y,w);
    return 0;
}
```

```
int GCD (int x, int y) // greatest common divisor
{
    int r, z;
    r = x % y;          //  x / y = c*y + r
    // {F: x>0 and y>0}
    if (r == 0)  z = y; else z = GCD (y,r);
    // {G: z = GCD (x,y)}
    return z;
}
```