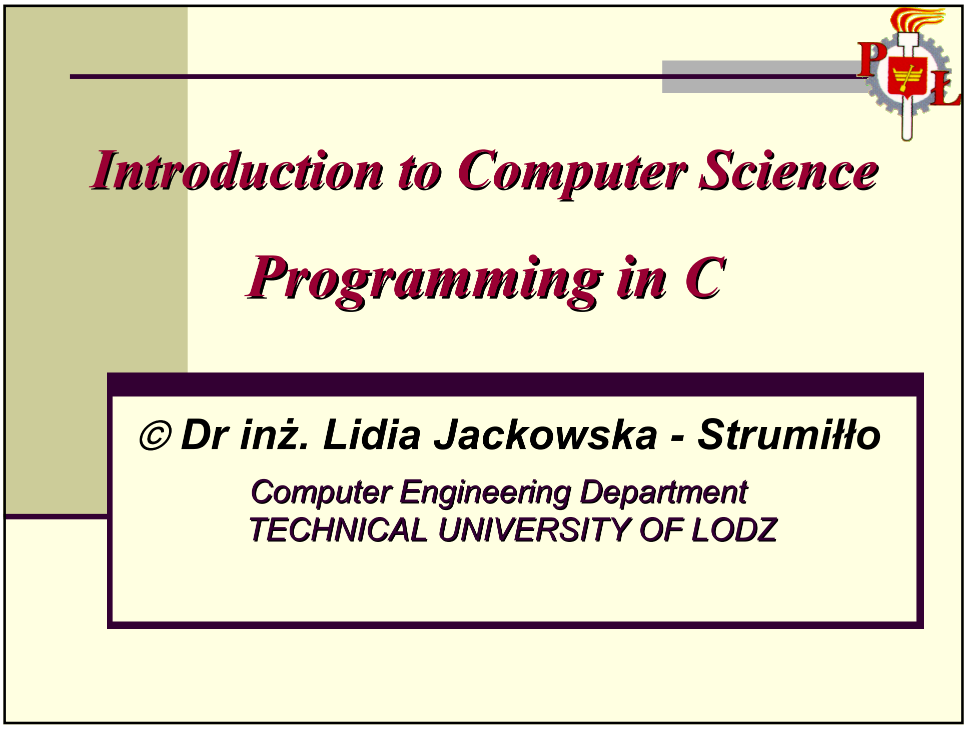
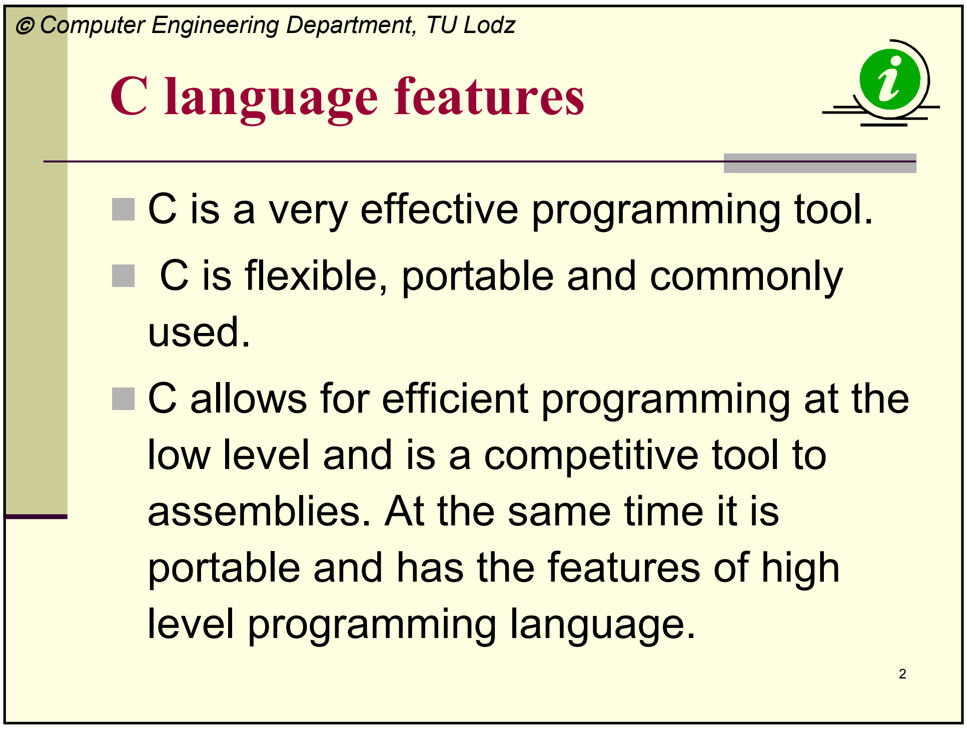


Introduction to Computer Science

Programming in C

© Dr inż. Lidia Jackowska - Strumiłło

Computer Engineering Department
TECHNICAL UNIVERSITY OF LODZ



C language features

- C is a very effective programming tool.
- C is flexible, portable and commonly used.
- C allows for efficient programming at the low level and is a competitive tool to assemblies. At the same time it is portable and has the features of high level programming language.

History

C language was defined at the begin of seventies by two employs of AT&T Bell Laboratories:

Brian Kernighan and Dennis Ritchie.

C was used to write a code of UNIX operating system.

Important dates:

1978 – a book *„C language”* was published,

1989 – ANSI C standard,

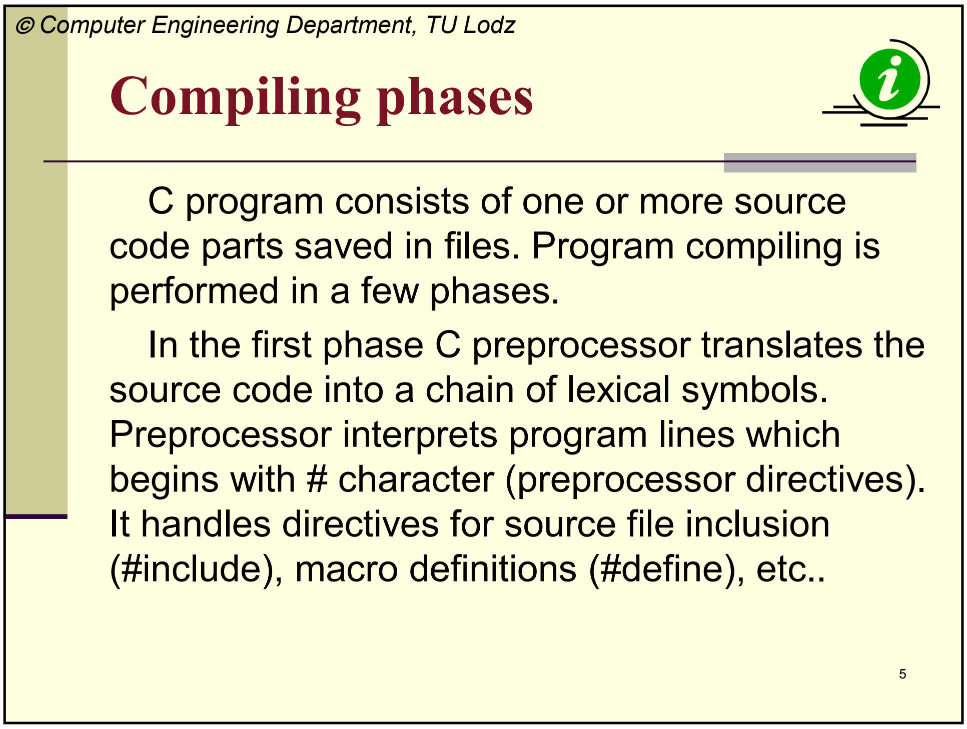
1990 – ISO 9899:1990 C standard.

Basic symbols

C alphabet consists of:

- small and capital letters of Latin alphabet and “_” (underline character),
- arabic numerals,
- special characters:

  + - * / = < > ( ) [ ] { } . , : ; ‘ ‘ " " ^ ! # & % | ~ ?

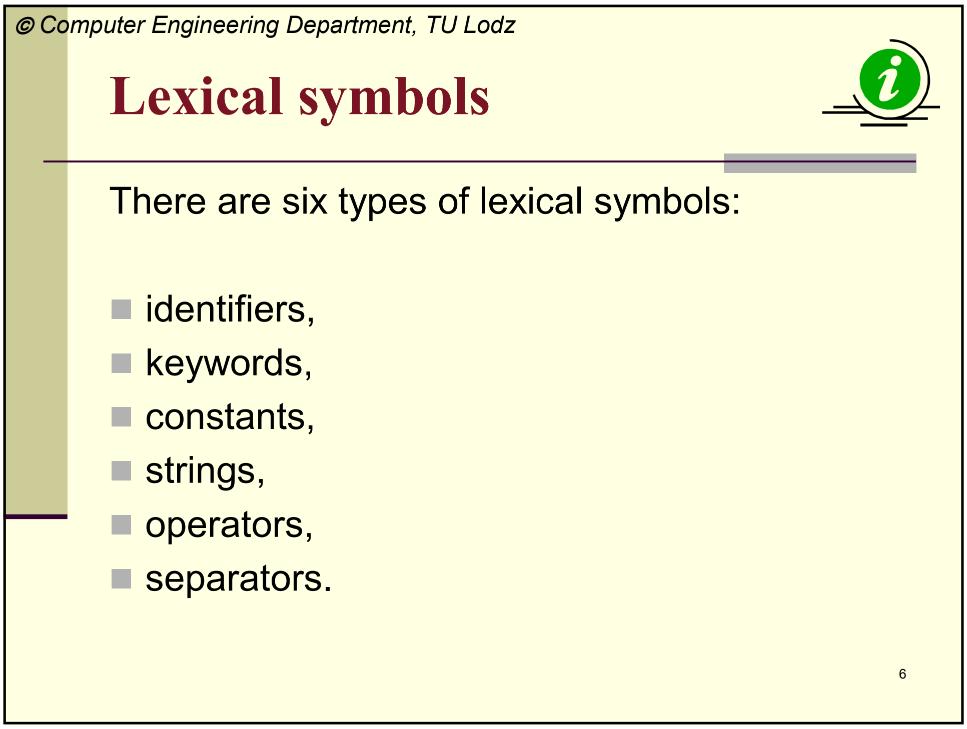


© Computer Engineering Department, TU Lodz

Compiling phases

C program consists of one or more source code parts saved in files. Program compiling is performed in a few phases.

In the first phase C preprocessor translates the source code into a chain of lexical symbols. Preprocessor interprets program lines which begins with # character (preprocessor directives). It handles directives for source file inclusion (#include), macro definitions (#define), etc..



© Computer Engineering Department, TU Lodz

Lexical symbols

There are six types of lexical symbols:

- identifiers,
- keywords,
- constants,
- strings,
- operators,
- separators.

Identifiers

Identifiers are chains of alphanumeric characters (letters and digits) and the first letter of the data name must be ALPHABETIC (i.e. A to Z or a to z). Lowercase and uppercase characters are differentiated.

A number of identifier characters is not limited.

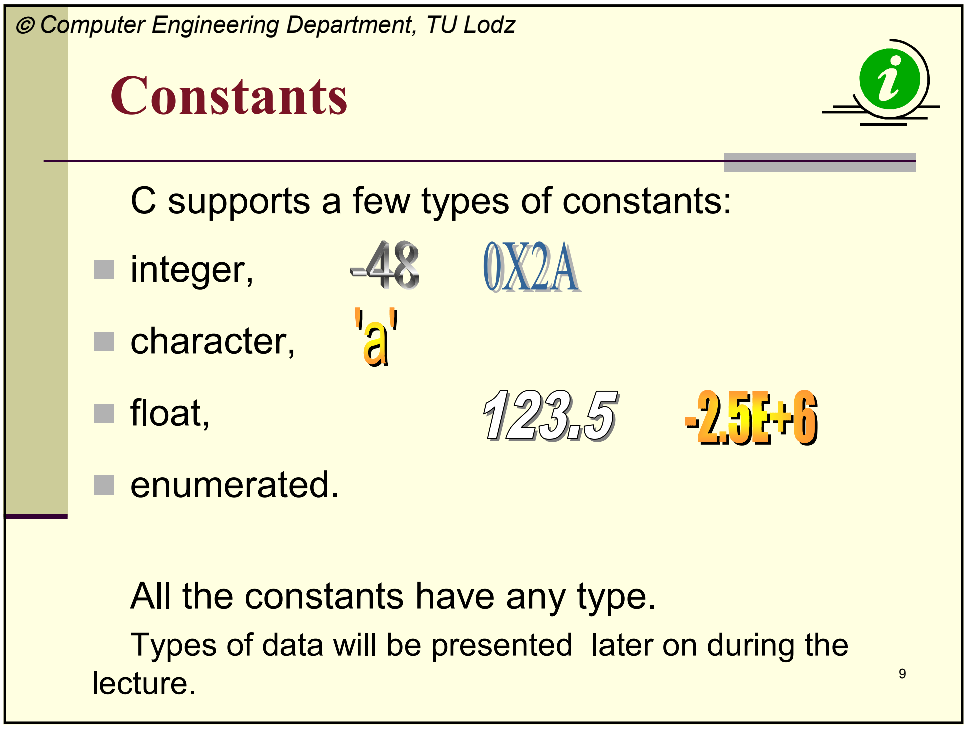
At least first 31 characters of **internal identifier** has its meaning, but in some implementations their number can be upper.

External identifiers are more limited, for example to first six characters (lowercase and uppercase characters are not differentiated) in some applications.

Keywords

auto	break	case	char	const	continue	default	do
double	else	enum	extern	float	for	goto	if
int	long	register	return	short	signed	sizeof	static
struct	switch	typedef	union	unsigned	void	volatile	while

Keywords must be written in lower case



© Computer Engineering Department, TU Lodz

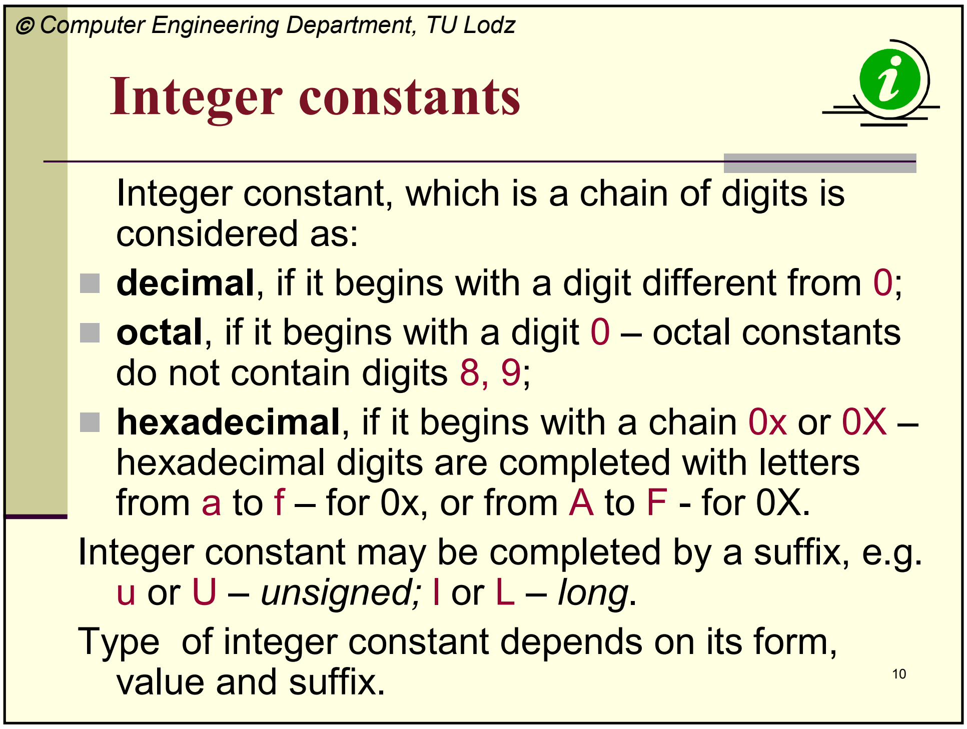
Constants

C supports a few types of constants:

- integer, -48 0X2A
- character, 'a'
- float, 123.5 -2.5E+6
- enumerated.

All the constants have any type.

Types of data will be presented later on during the lecture.



© Computer Engineering Department, TU Lodz

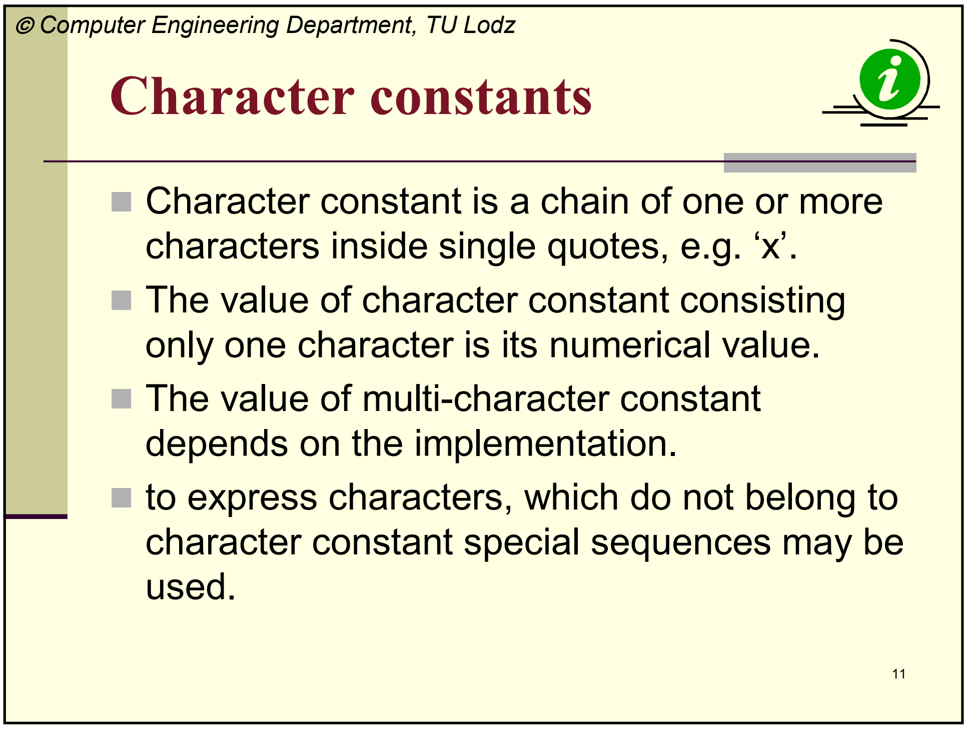
Integer constants

Integer constant, which is a chain of digits is considered as:

- **decimal**, if it begins with a digit different from 0;
- **octal**, if it begins with a digit 0 – octal constants do not contain digits 8, 9;
- **hexadecimal**, if it begins with a chain 0x or 0X – hexadecimal digits are completed with letters from a to f – for 0x, or from A to F - for 0X.

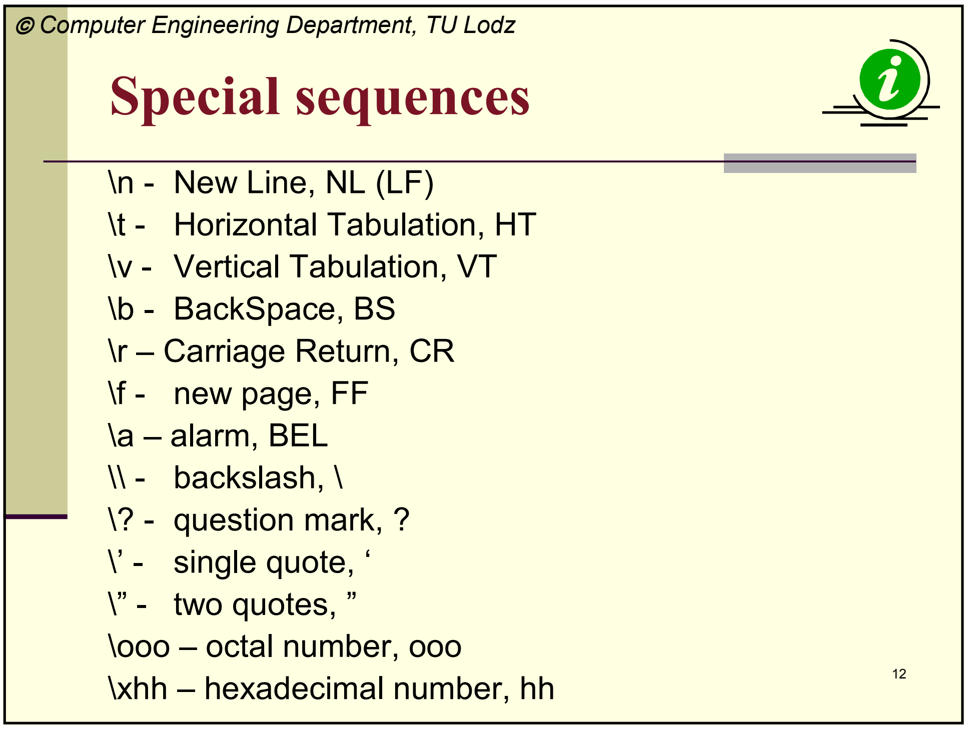
Integer constant may be completed by a suffix, e.g. u or U – *unsigned;* l or L – *long*.

Type of integer constant depends on its form, value and suffix.



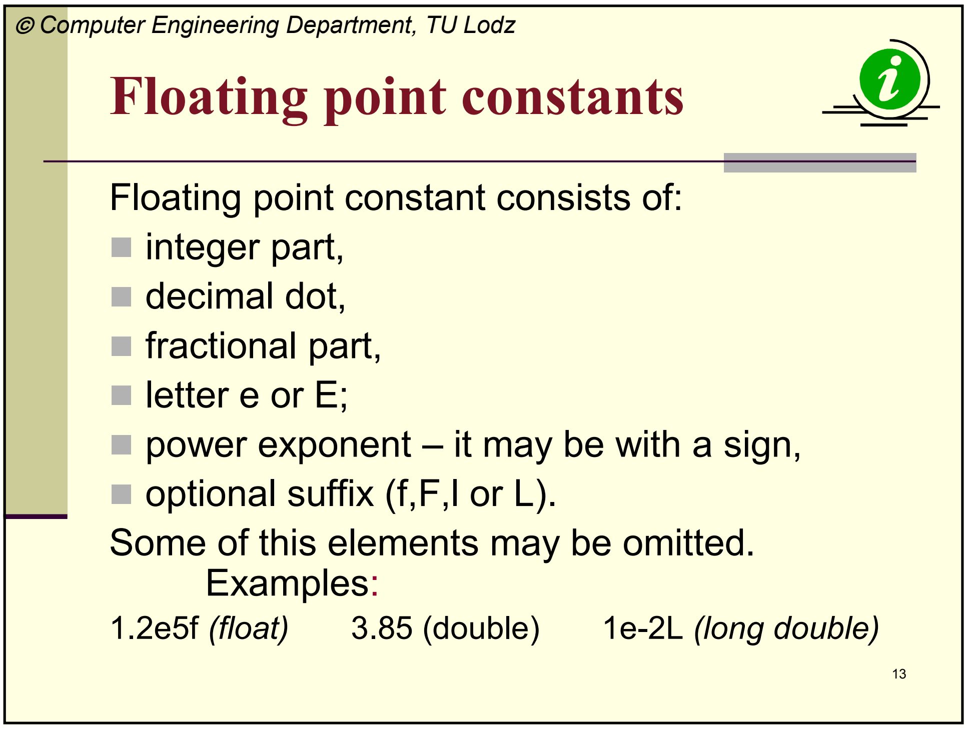
Character constants

- Character constant is a chain of one or more characters inside single quotes, e.g. ‘x’.
- The value of character constant consisting only one character is its numerical value.
- The value of multi-character constant depends on the implementation.
- to express characters, which do not belong to character constant special sequences may be used.



Special sequences

\n - New Line, NL (LF)
\t - Horizontal Tabulation, HT
\v - Vertical Tabulation, VT
\b - BackSpace, BS
\r – Carriage Return, CR
\f - new page, FF
\a – alarm, BEL
\\ - backslash, \
\? - question mark, ?
\’ - single quote, ‘
\” - two quotes, ”
\ooo – octal number, ooo
\xhh – hexadecimal number, hh



Floating point constants

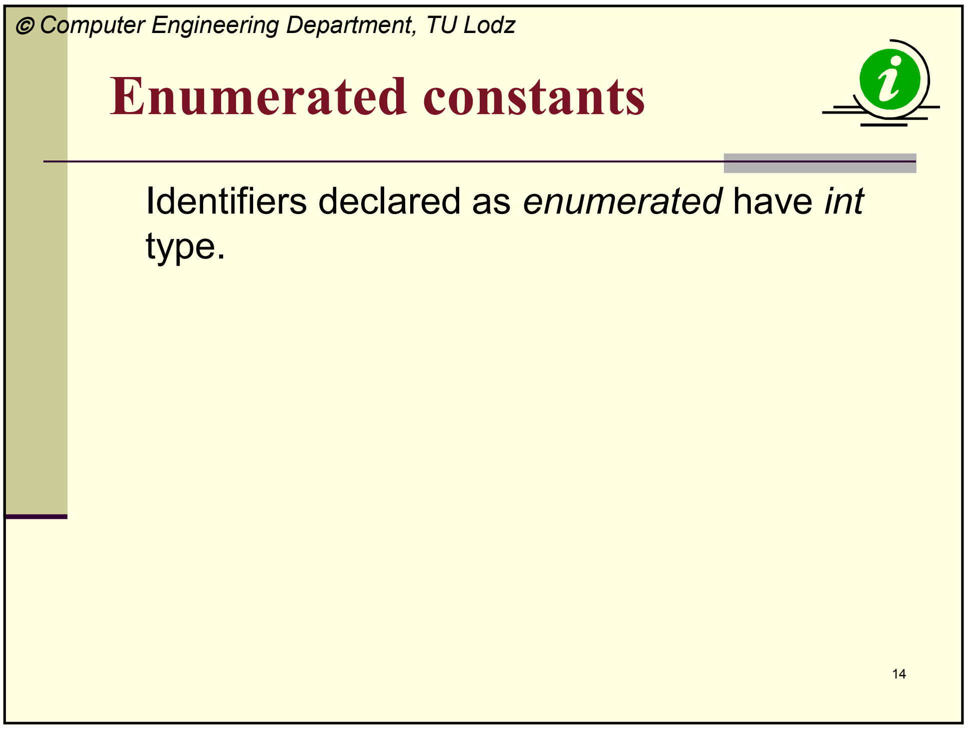
Floating point constant consists of:

- integer part,
- decimal dot,
- fractional part,
- letter e or E;
- power exponent – it may be with a sign,
- optional suffix (f,F,l or L).

Some of this elements may be omitted.

Examples:

1.2e5f *(float)* 3.85 (double) 1e-2L *(long double)*



Enumerated constants

Identifiers declared as *enumerated* have *int* type.

Character strings

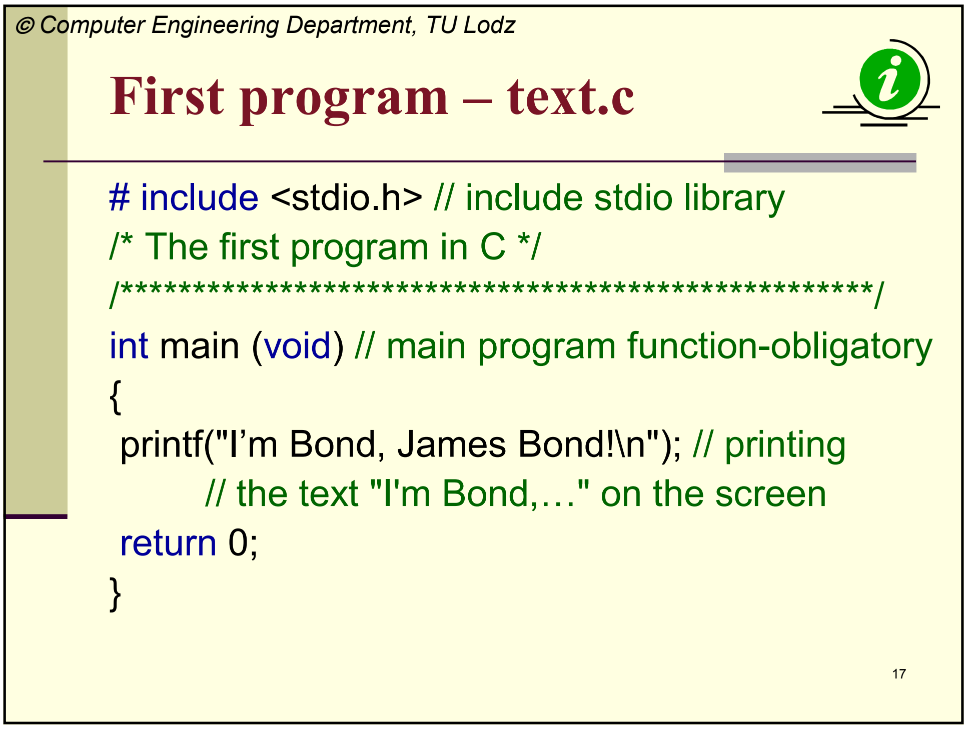
- **String** is a chain of basic symbols and other ASCII characters inside two quotes, e.g. "abc".
- The same special sequences as in character constants may be used in character strings.
- Neighbouring strings may be concatenated during the program compiling.
- String constant is an array, which elements are characters. The last element of array is character '\0', which do not belong to string.

Separators

- **Separators** are:
  - comments – chains of characters within /* and */ or after // until the end of line - comments within /*… */ mustn't be inside other comment, string or constant;

    Comments are ignored by the compiler, but are helpful to explain how the program works to the programmers.
  - spaces,
  - tabulators,
  - end of line or page characters, etc.

Separators are called *„white characters”.*

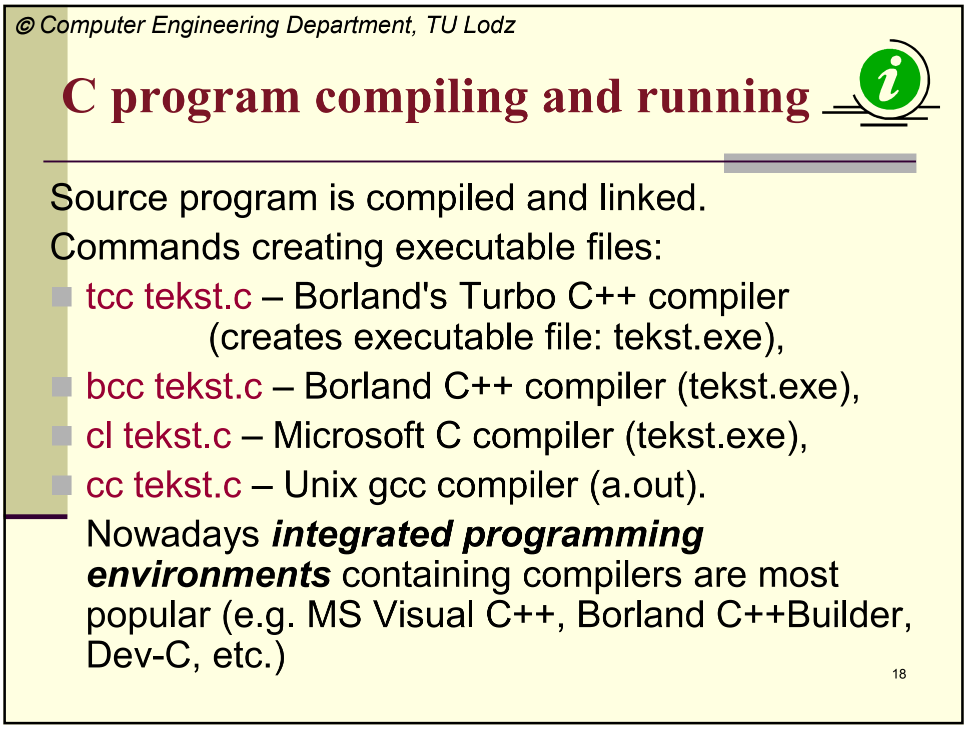


© Computer Engineering Department, TU Lodz

First program – text.c

```
# include <stdio.h> // include stdio library
/* The first program in C */
/************************************************/
int main (void) // main program function-obligatory
{
 printf("I’m Bond, James Bond!\n"); // printing
         // the text "I'm Bond,…" on the screen
 return 0;
}
```

17



© Computer Engineering Department, TU Lodz

C program compiling and running

Source program is compiled and linked.

Commands creating executable files:

- tcc tekst.c – Borland's Turbo C++ compiler (creates executable file: tekst.exe),
- bcc tekst.c – Borland C++ compiler (tekst.exe),
- cl tekst.c – Microsoft C compiler (tekst.exe),
- cc tekst.c – Unix gcc compiler (a.out).

Nowadays ***integrated programming environments*** containing compilers are most popular (e.g. MS Visual C++, Borland C++Builder, Dev-C, etc.)

18

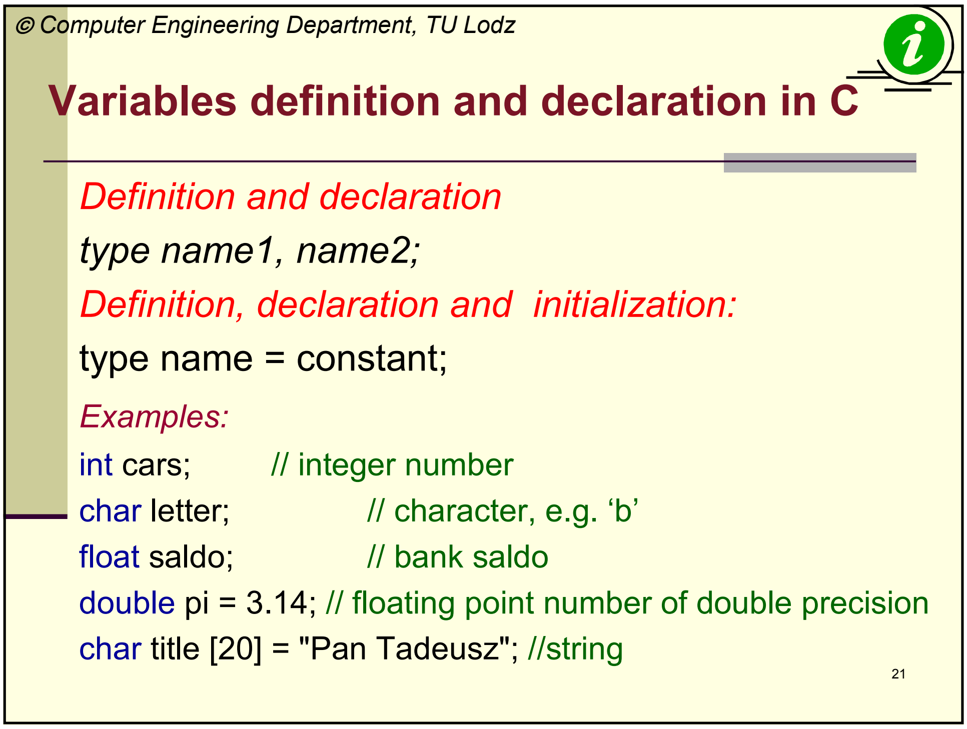
Basic data types

There are four basic data types in C :

- **char** – a single byte storing one character,
- **int** – integer number, its size depends on the system,
- **float** – floating point number of single precision,
- **double** - floating point number of double precision.

Variables

- ***Variable*** can change its value within the declared type.
- ***Variables*** *store values and information. They allow programs to perform calculations and store data for later retrieval.*
- *Variables store numbers, names, text messages, etc*
- All *variables* must be declared before the first usage in program.



Variables definition and declaration in C

Definition and declaration

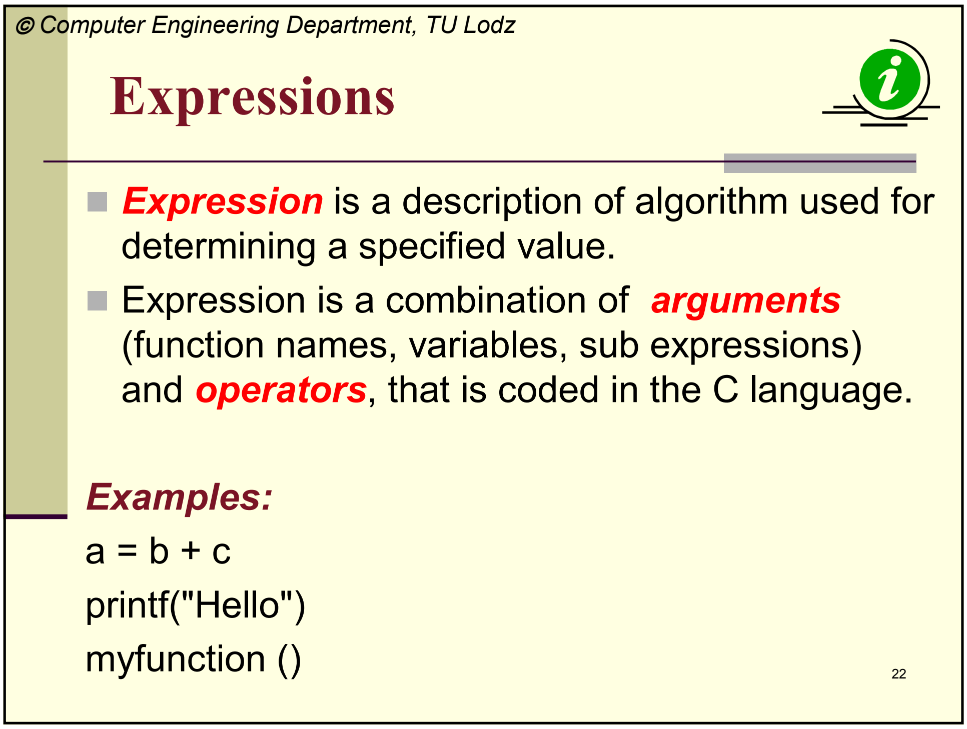
type name1, name2;

Definition, declaration and initialization:

type name = constant;

Examples:

```
int cars;         // integer number
char letter;            // character, e.g. ‘b’
float saldo;            // bank saldo
double pi = 3.14; // floating point number of double precision
char title [20] = "Pan Tadeusz"; //string
```



Expressions

- ***Expression*** is a description of algorithm used for determining a specified value.
- Expression is a combination of ***arguments*** (function names, variables, sub expressions) and ***operators***, that is coded in the C language.

Examples:

```
a = b + c
printf("Hello")
myfunction ()
```

Instructions (statements)

- ***Instructions*** are language statements describing operations on the data.
- There are two kinds of instructions in C:
  - ***simple***, which do not contain other instructions as their components,
  - ***structural***, which might be extended to the structure of multiple statements.
- Simple instruction (statement) is an expression finished with semicolon.

Expressions and statements

Examples:

Expressions:	*Simple instructions:*
a = b + 0.5*c	a = b + 0.5*c; // assignment
printf("Hello")	printf("Hello");
myfunction()	myfunction(); //function call

Simple statements

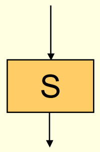
- ***Assignment:***

  z = w;

- ***Function call :***

  function_name (*list of arguments*);

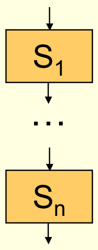
Statement symbol in algorithm flowchart:



Block of statements

```
{
    statement_1
    statement_ 2
    …
    statement_n
}
```

Block symbol in algorithm flowchart:



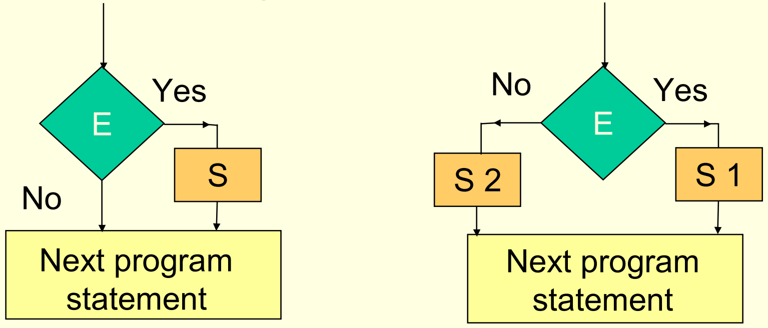
Structural instructions

♦ **Conditional *if* and *if-else* statements:**

if (expression) statement

if (expression) statement1 **else** statement2

Conditions in algorithm flowcharts:

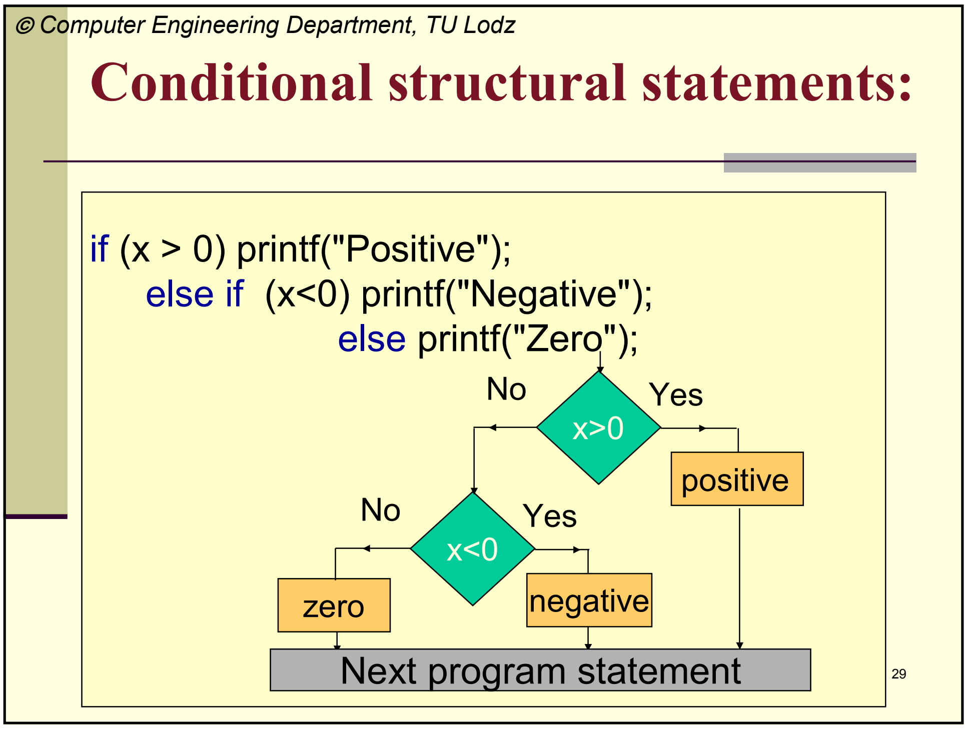


If statements

if (expression) statement1 [else statement2]

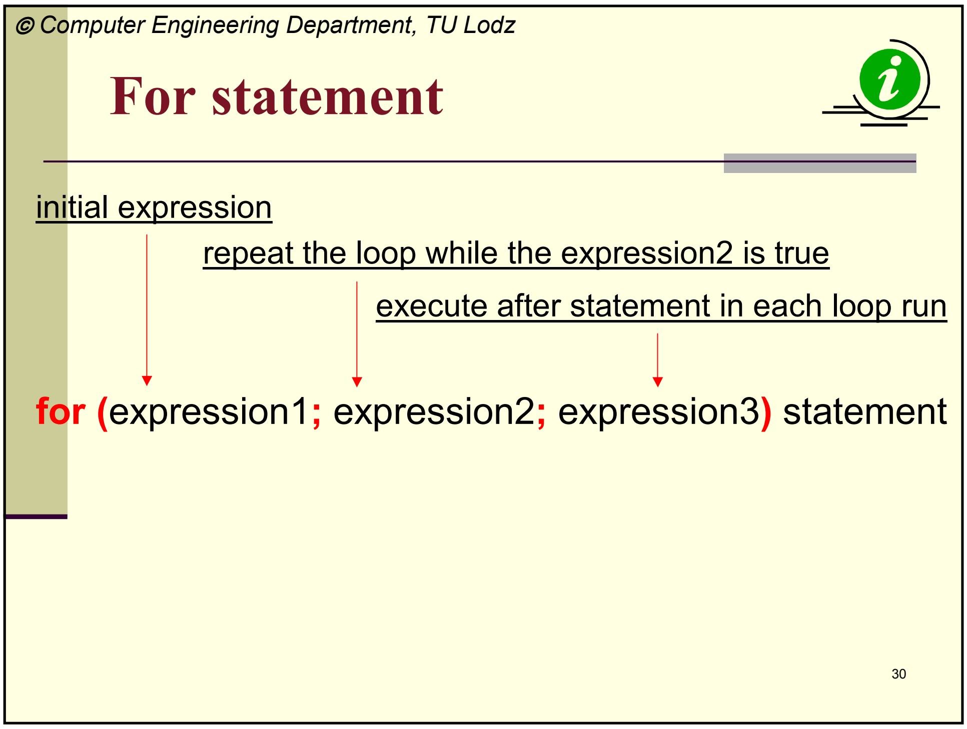
Examples:

```
if (s == 2) printf("Two solutions");
if (x > y)  a = 1;   else a = 0;
if (w == z)
    {
            a=z-32.5;
            printf("Expression solution: %f\n", a);
    }
```



Conditional structural statements:

```
if (x > 0) printf("Positive");
    else if  (x<0) printf("Negative");
                 else printf("Zero");
```



For statement

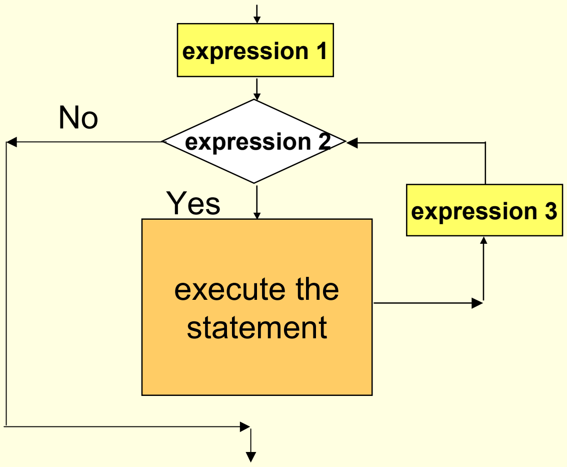
initial expression
repeat the loop while the expression2 is true
execute after statement in each loop run

```
for (expression1; expression2; expression3) statement
```

For statement

for (expression1**;** expression2**;** expression3**)** statement

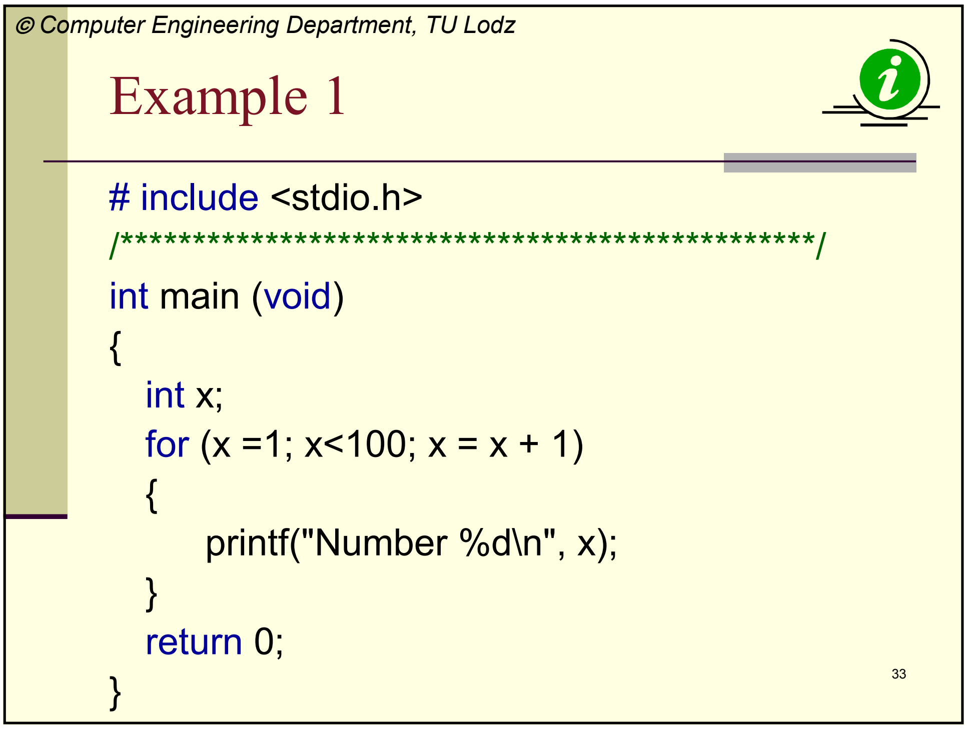
Loop in algorithm flowchart:



Examples:

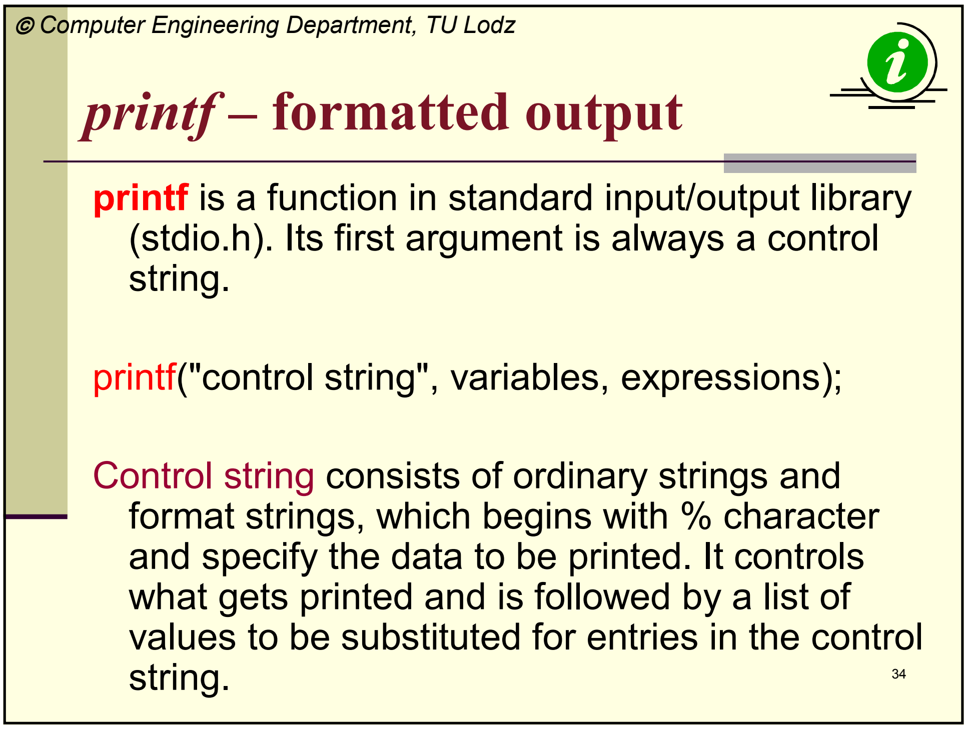
```
for (i=1; i<10; i++) printf(”i =%d”, i);

for (i=10; i>1; i--)
{
        x=5+10*i;
        printf(”i=%d, x=%d”, i, x);
}
```



Example 1

```
# include <stdio.h>
/*************************************************/
int main (void)
{
   int x;
   for (x =1; x<100; x = x + 1)
   {
        printf("Number %d\n", x);
   }
   return 0;
}
```

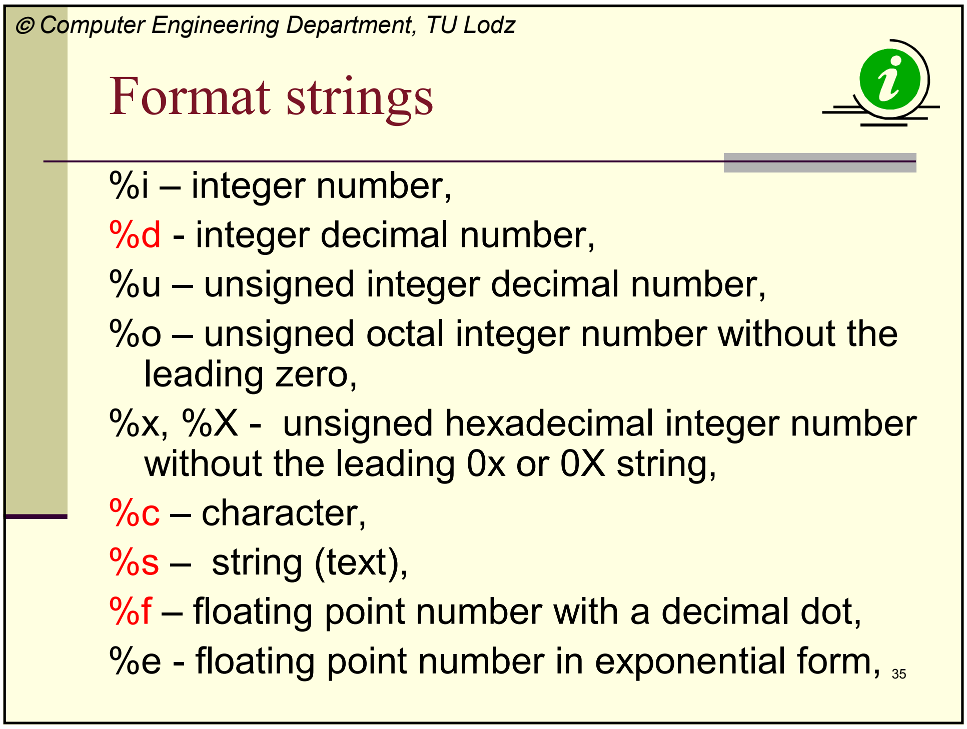


printf – formatted output

printf is a function in standard input/output library (stdio.h). Its first argument is always a control string.

printf("control string", variables, expressions);

Control string consists of ordinary strings and format strings, which begins with % character and specify the data to be printed. It controls what gets printed and is followed by a list of values to be substituted for entries in the control string.



Format strings

%i – integer number,

%d - integer decimal number,

%u – unsigned integer decimal number,

%o – unsigned octal integer number without the leading zero,

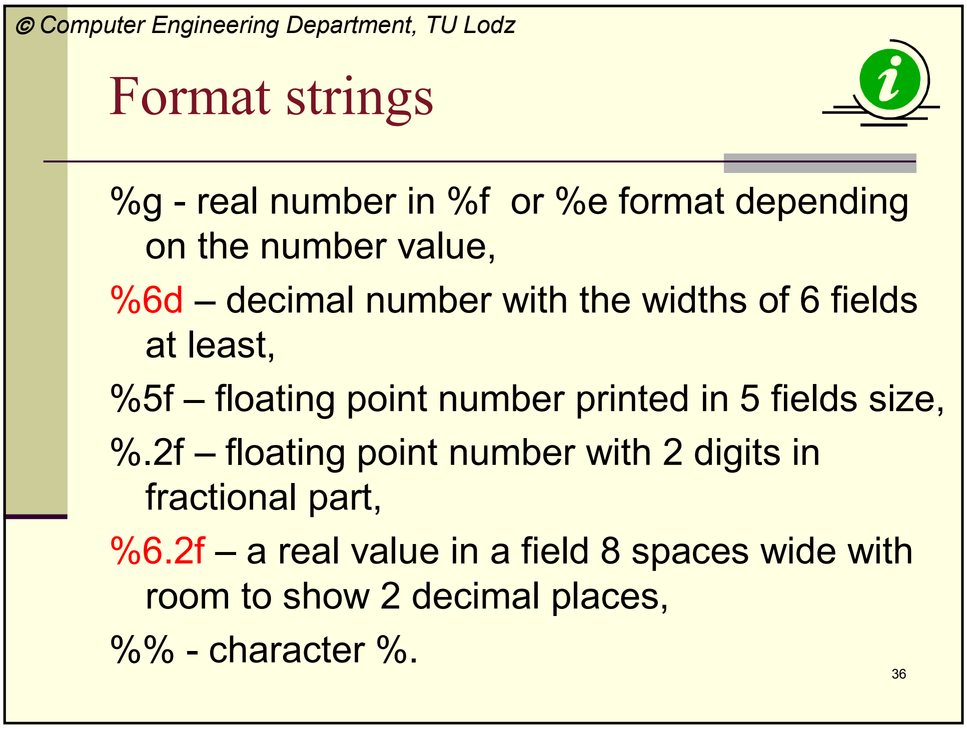
%x, %X - unsigned hexadecimal integer number without the leading 0x or 0X string,

%c – character,

%s – string (text),

%f – floating point number with a decimal dot,

%e - floating point number in exponential form,



Format strings

%g - real number in %f or %e format depending on the number value,

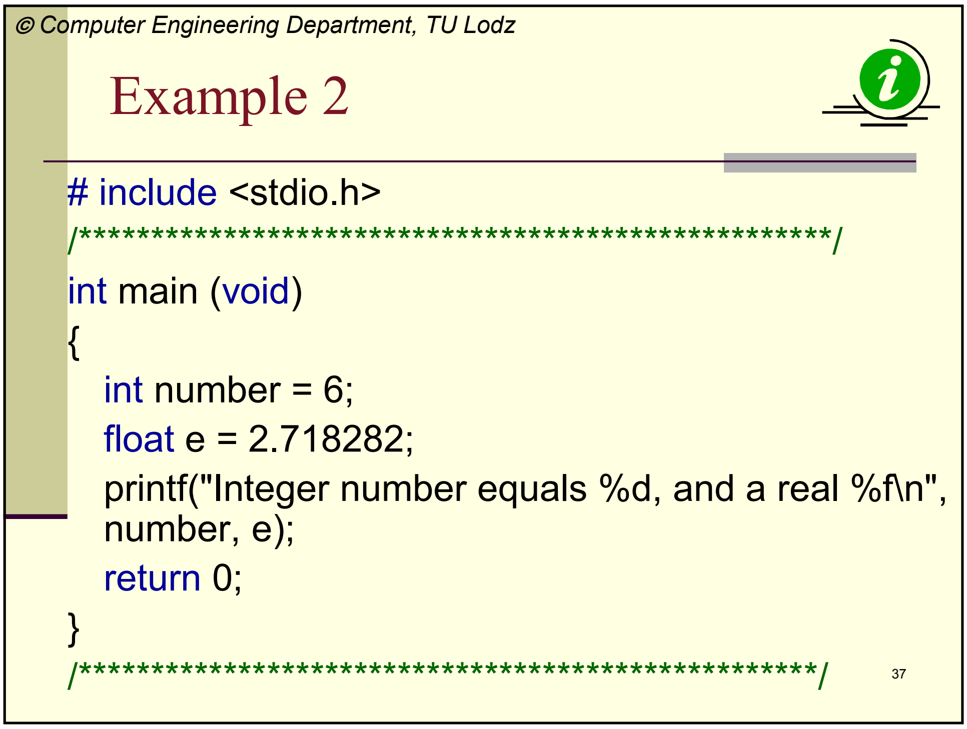
%6d – decimal number with the widths of 6 fields at least,

%5f – floating point number printed in 5 fields size,

%.2f – floating point number with 2 digits in fractional part,

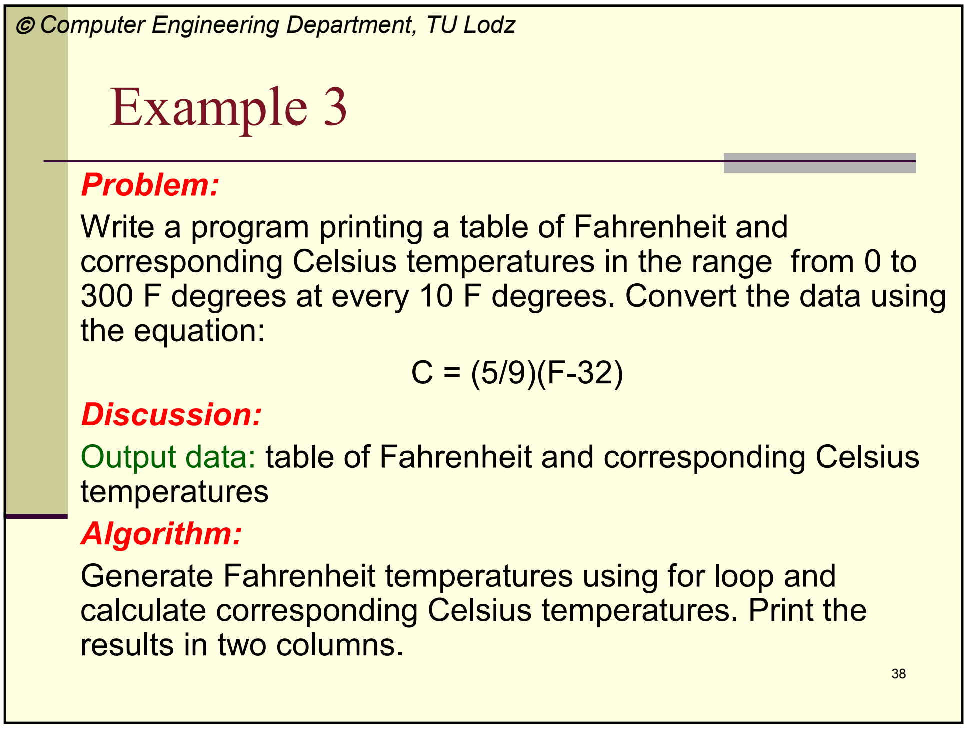
%6.2f – a real value in a field 8 spaces wide with room to show 2 decimal places,

%% - character %.



Example 2

```
# include <stdio.h>
/**************************************************/
int main (void)
{
   int number = 6;
   float e = 2.718282;
   printf("Integer number equals %d, and a real %f\n",
   number, e);
   return 0;
}
/**************************************************/
```



Example 3

Problem:

Write a program printing a table of Fahrenheit and corresponding Celsius temperatures in the range from 0 to 300 F degrees at every 10 F degrees. Convert the data using the equation:

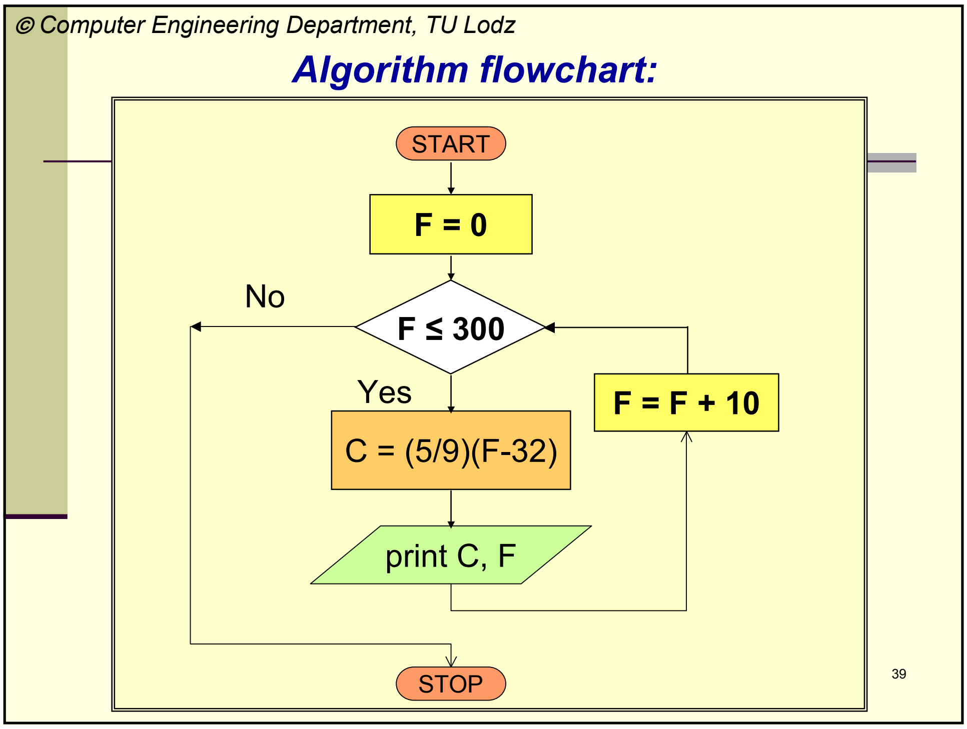
$$C = (5/9)(F-32)$$

Discussion:

Output data: table of Fahrenheit and corresponding Celsius temperatures

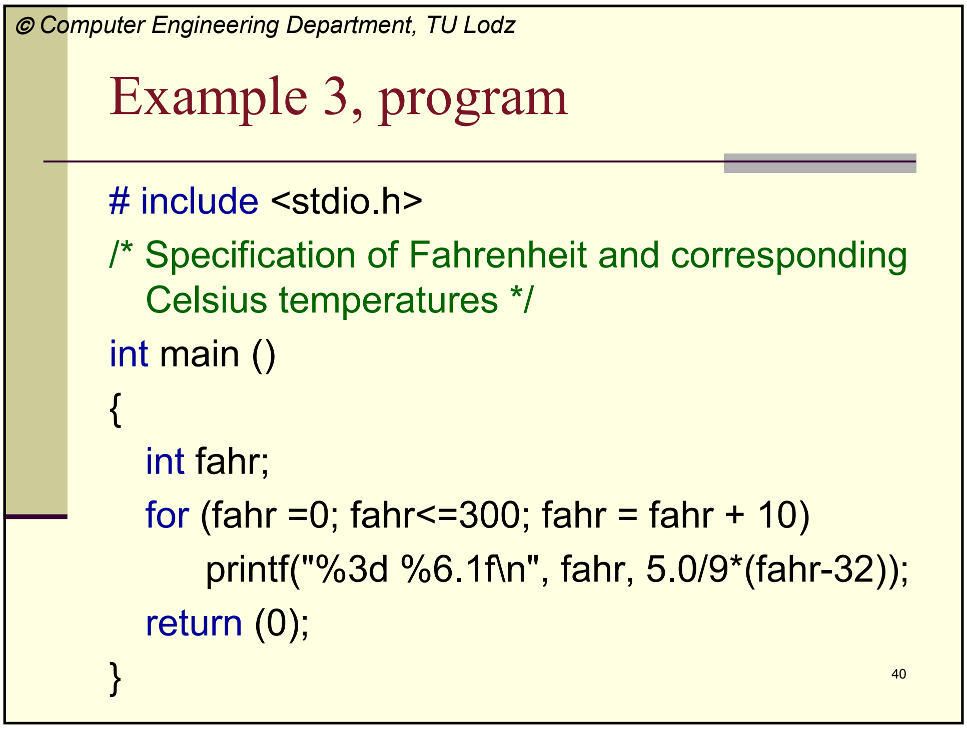
Algorithm:

Generate Fahrenheit temperatures using for loop and calculate corresponding Celsius temperatures. Print the results in two columns.



© Computer Engineering Department, TU Lodz

Algorithm flowchart:



© Computer Engineering Department, TU Lodz

Example 3, program

```
# include <stdio.h>
/* Specification of Fahrenheit and corresponding
   Celsius temperatures */
int main ()
{
   int fahr;
   for (fahr =0; fahr<=300; fahr = fahr + 10)
        printf("%3d %6.1f\n", fahr, 5.0/9*(fahr-32));
   return (0);
}
```

Example 4

Write a program that calculates an area of a circle for any radius entered by the user.

Discussion:

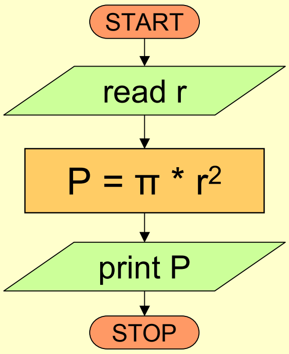
Input data: radius

Output data: area

Algorithm:

$$P = \pi r^2$$

Algorithm flowchart



Program example 1

```
/*File: circle_1.c      Program calculates an area of a circle*/

#include <stdio.h>
#define PI 3.14159                 /*declarations*/

int main()                 /* main program – executive part */
{
   float r,a;                      /* radius, area */
   printf("Enter radius: ");
   scanf("%f", &r);                /* read radius*/
   a=PI*r*r;                       /* compute circle area */
   printf(" An area of a circle for the radius r = %.4f equals
   %.4f", r, a);                           /* write the result */
   return 0;
}
```

Program clarity

```
#include <stdio.h>
#define PI 3.14159
int main()
{
    float r,area;
    scanf("%f", &r);
    area=PI*r*r;
    printf(" An area of a circle for the radius r = %.4f equals %.4f", r, area);
    return 0;
}
```

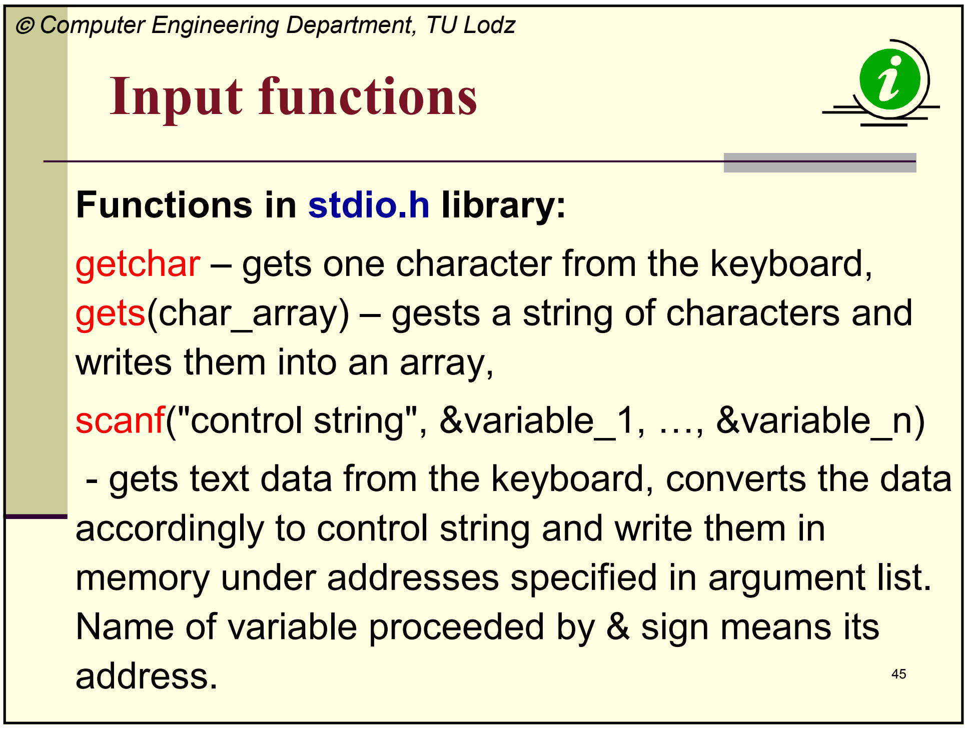
```
#include <stdio.h> #define PI 3.14159
int main() { float r, area; scanf("%f", &r);
area=PI*r*r; printf(" An area of a circle for the radius r = %.4f equals %.4f",
    r, area);return 0;}
```

Program example 2

```
/*File circle_2.c                   Program calculates an area of a circle */

#include <stdio.h>
#include <math.h>

int main ()                                /* main program */
{
    float radius,area;                     /*variables declarations */
    printf(" 'Program calculates an area of a circle \n");
    printf ("Enter the radius value, r =" );
    scanf("%f",&radius);                           /*read radius*/
    area=M_PI*pow(radius,2);                       /* compute circle area */
    printf(" An area of a circle for the radius r = %.4f equals %.4f", radius,
    area);                                         /*write the result*/
    return 0;
}
```



Input functions

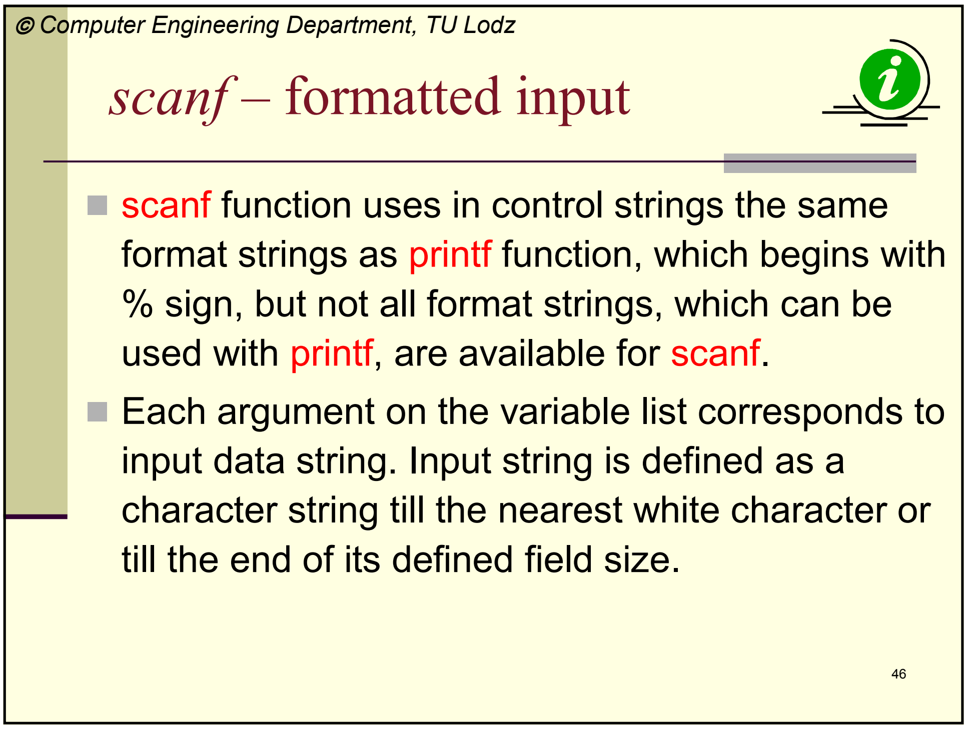
Functions in stdio.h library:

getchar – gets one character from the keyboard,

gets(char_array) – gests a string of characters and writes them into an array,

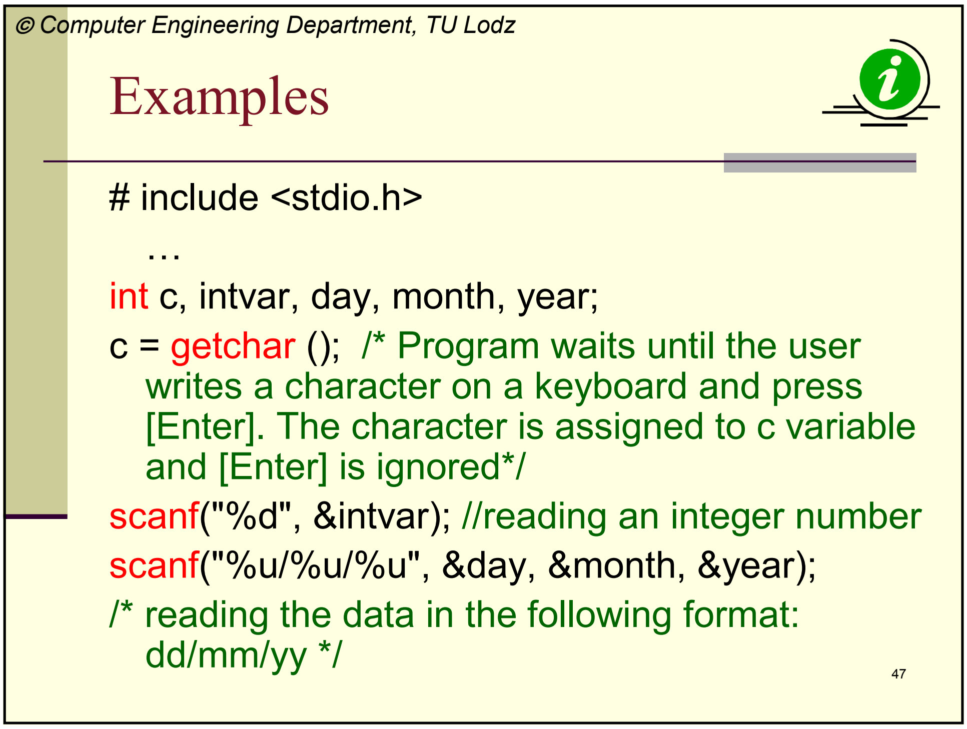
scanf("control string", &variable_1, …, &variable_n)

- gets text data from the keyboard, converts the data accordingly to control string and write them in memory under addresses specified in argument list. Name of variable proceeded by & sign means its address.



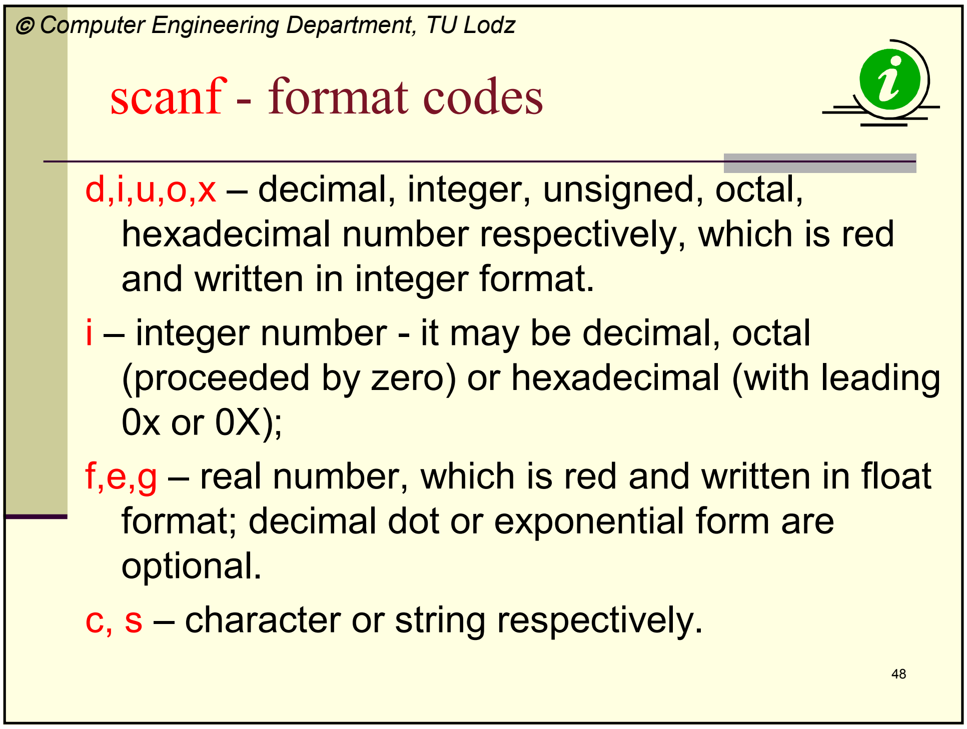
scanf – formatted input

- scanf function uses in control strings the same format strings as printf function, which begins with % sign, but not all format strings, which can be used with printf, are available for scanf.
- Each argument on the variable list corresponds to input data string. Input string is defined as a character string till the nearest white character or till the end of its defined field size.



Examples

```
# include <stdio.h>
   …
int c, intvar, day, month, year;
c = getchar ();  /* Program waits until the user
   writes a character on a keyboard and press
   [Enter]. The character is assigned to c variable
   and [Enter] is ignored*/
scanf("%d", &intvar); //reading an integer number
scanf("%u/%u/%u", &day, &month, &year);
/* reading the data in the following format:
   dd/mm/yy */
```



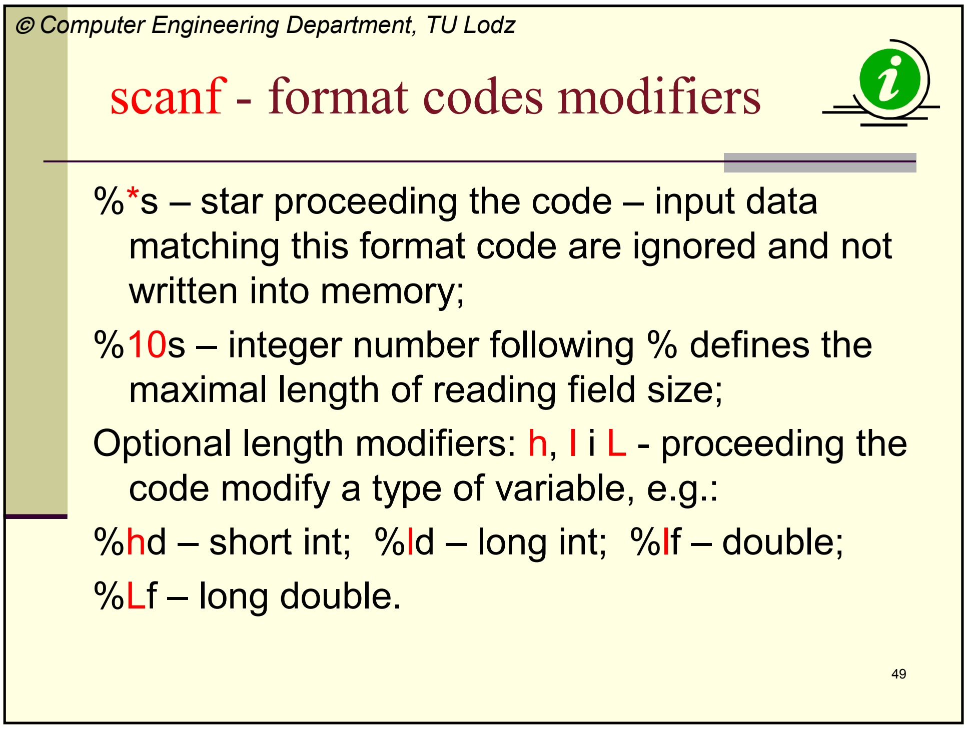
scanf - format codes

d,i,u,o,x – decimal, integer, unsigned, octal, hexadecimal number respectively, which is red and written in integer format.

i – integer number - it may be decimal, octal (proceeded by zero) or hexadecimal (with leading 0x or 0X);

f,e,g – real number, which is red and written in float format; decimal dot or exponential form are optional.

c, s – character or string respectively.



scanf - format codes modifiers

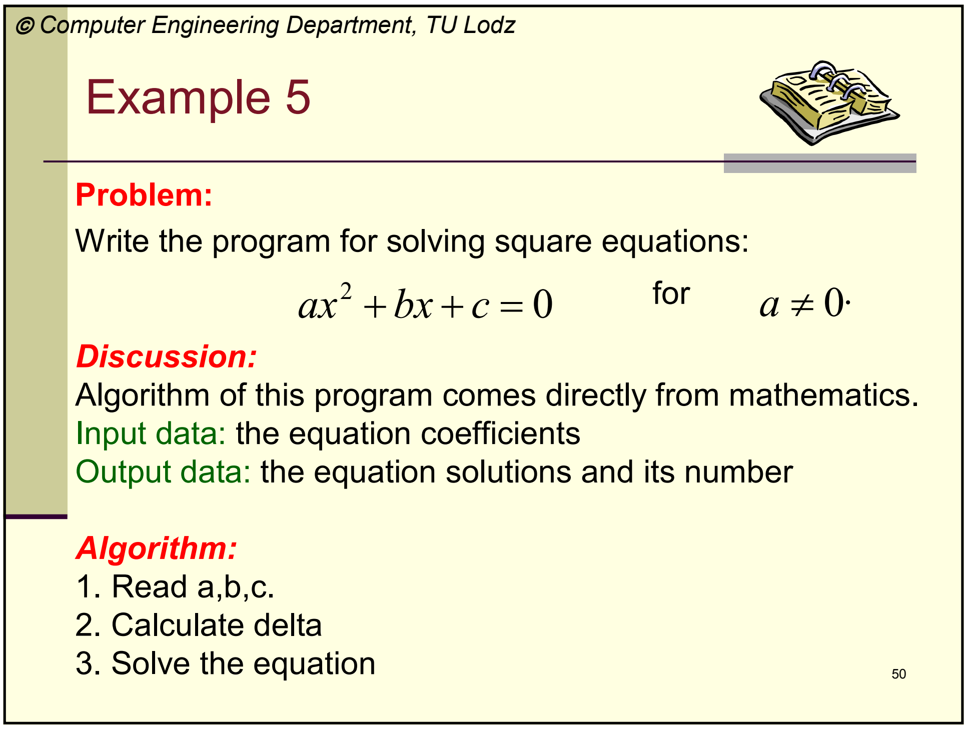
%*s – star proceeding the code – input data matching this format code are ignored and not written into memory;

%10s – integer number following % defines the maximal length of reading field size;

Optional length modifiers: h, l i L - proceeding the code modify a type of variable, e.g.:

%hd – short int; %ld – long int; %lf – double;

%Lf – long double.



Example 5

Problem:

Write the program for solving square equations:

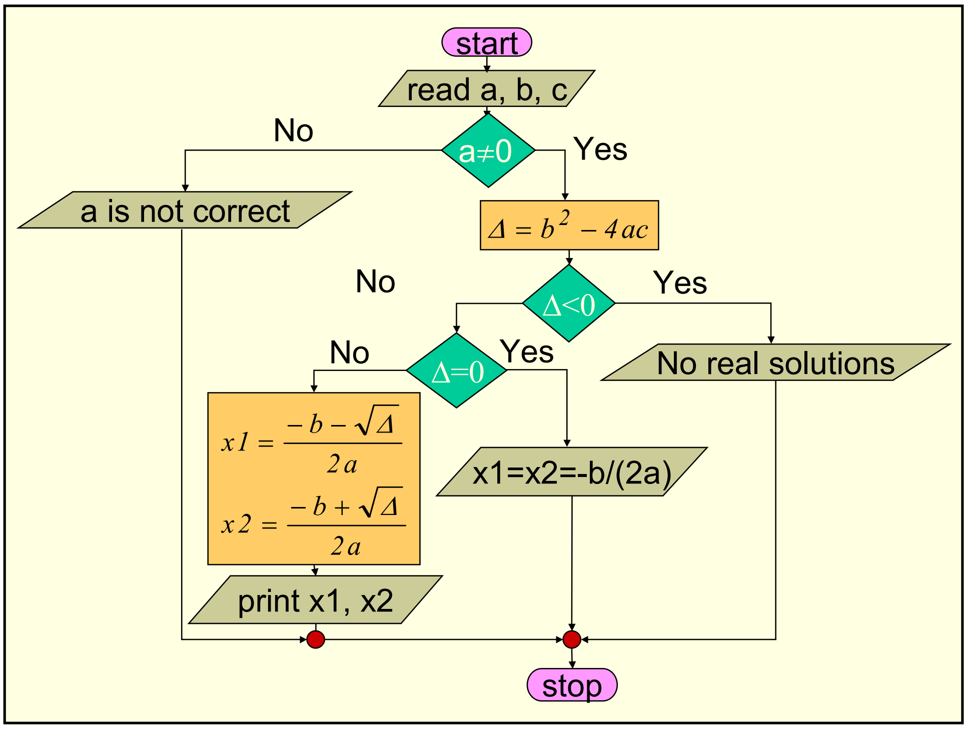
$$ax^2 + bx + c = 0 \qquad \text{for} \qquad a \neq 0.$$

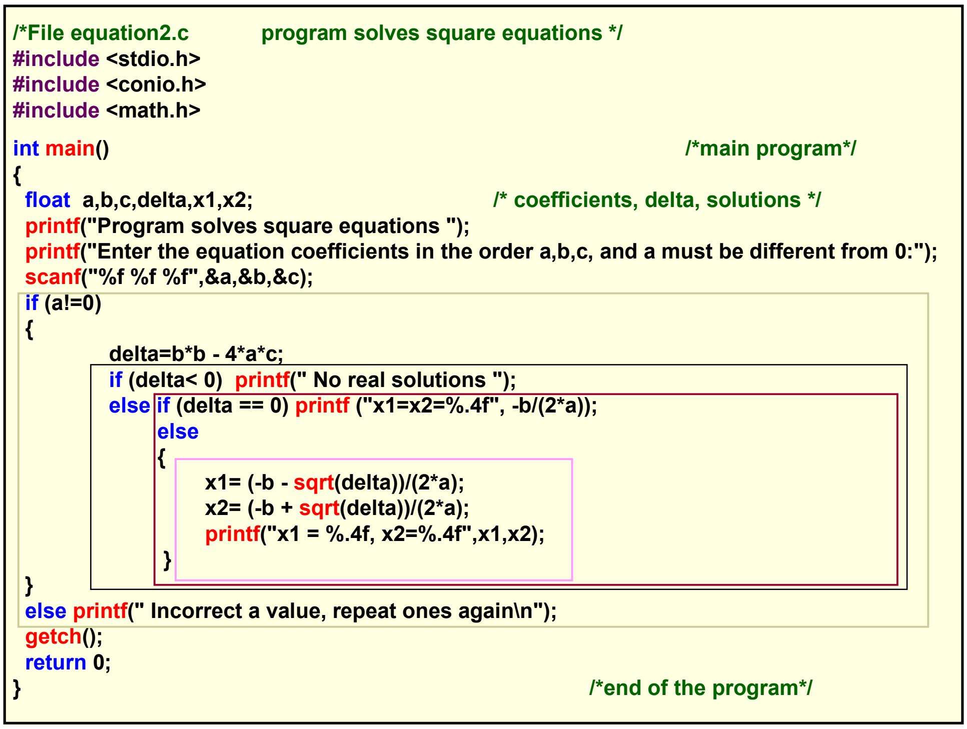
Discussion:

Algorithm of this program comes directly from mathematics.
Input data: the equation coefficients
Output data: the equation solutions and its number

Algorithm:

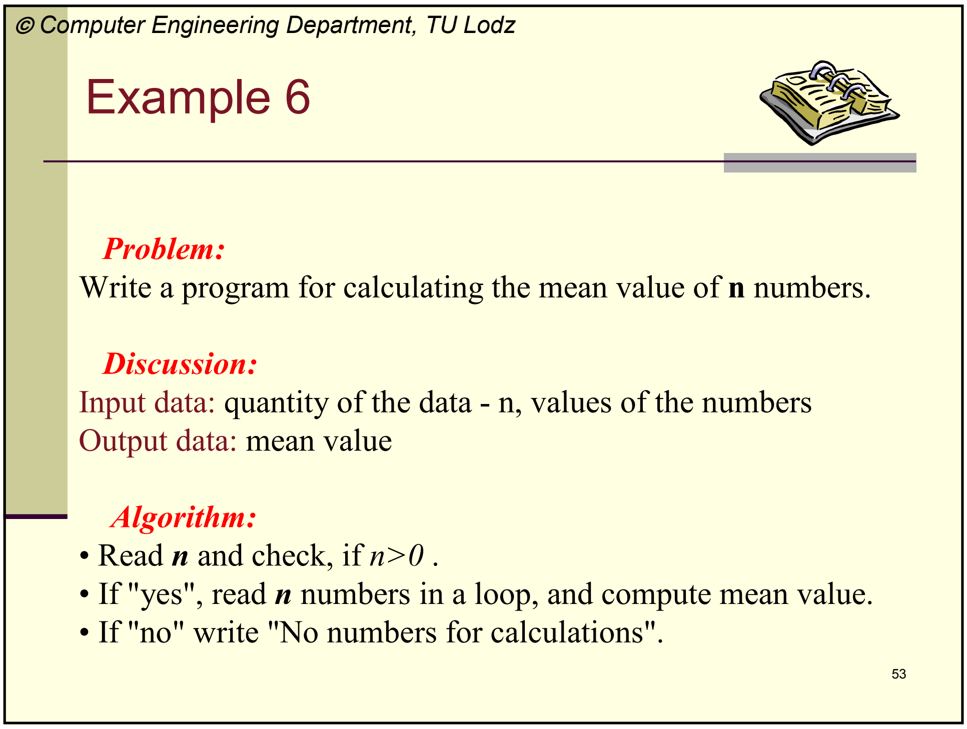
1. Read a,b,c.
2. Calculate delta
3. Solve the equation





```
/*File equation2.c          program solves square equations */
#include <stdio.h>
#include <conio.h>
#include <math.h>

int main()                                                 /*main program*/
{
  float  a,b,c,delta,x1,x2;                     /* coefficients, delta, solutions */
  printf("Program solves square equations ");
  printf("Enter the equation coefficients in the order a,b,c, and a must be different from 0:");
  scanf("%f %f %f",&a,&b,&c);
  if (a!=0)
  {
        delta=b*b - 4*a*c;
        if (delta< 0)  printf(" No real solutions ");
        else if (delta == 0) printf ("x1=x2=%.4f", -b/(2*a));
             else
             {
                 x1= (-b - sqrt(delta))/(2*a);
                 x2= (-b + sqrt(delta))/(2*a);
                 printf("x1 = %.4f, x2=%.4f",x1,x2);
             }
  }
  else printf(" Incorrect a value, repeat ones again\n");
  getch();
  return 0;
}                                                   /*end of the program*/
```



© Computer Engineering Department, TU Lodz

Example 6

Problem:

Write a program for calculating the mean value of **n** numbers.

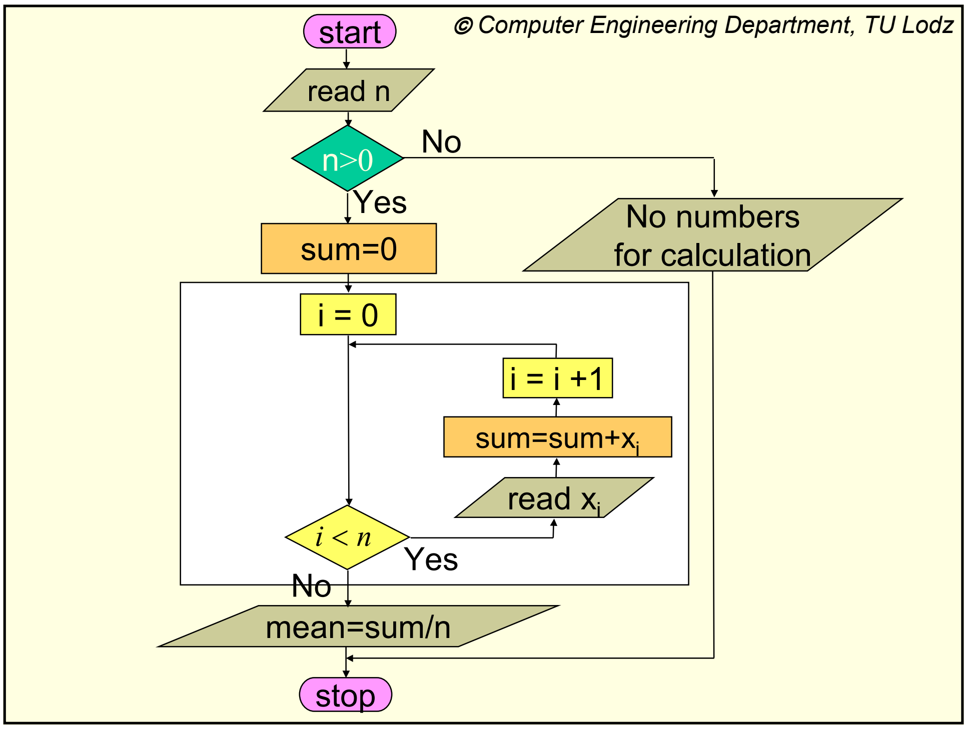
Discussion:

Input data: quantity of the data - n, values of the numbers

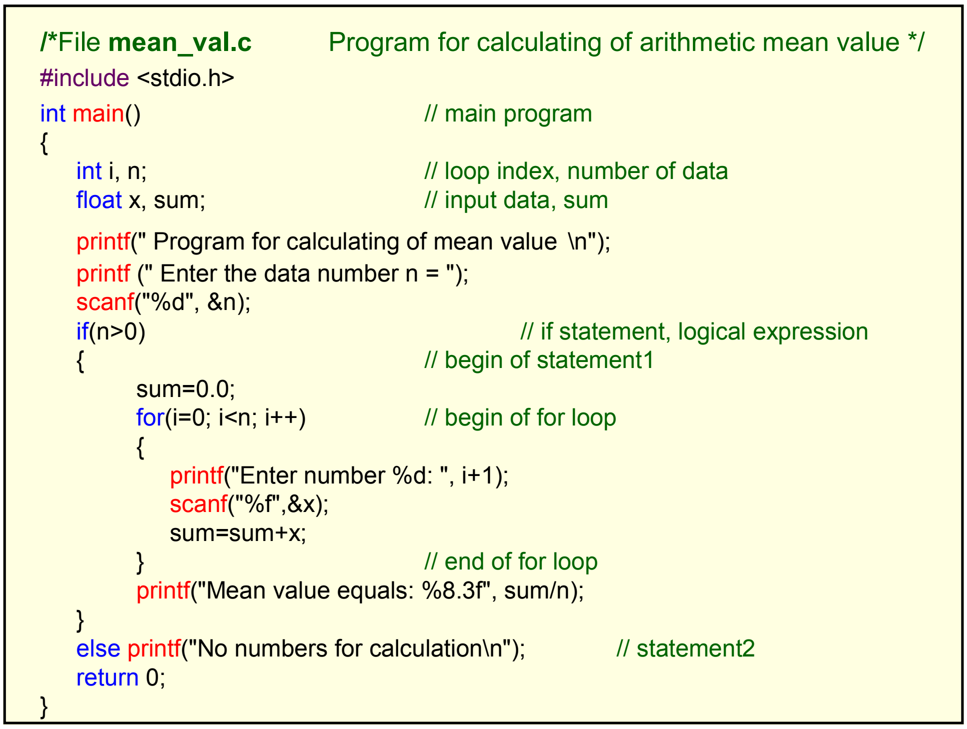
Output data: mean value

Algorithm:

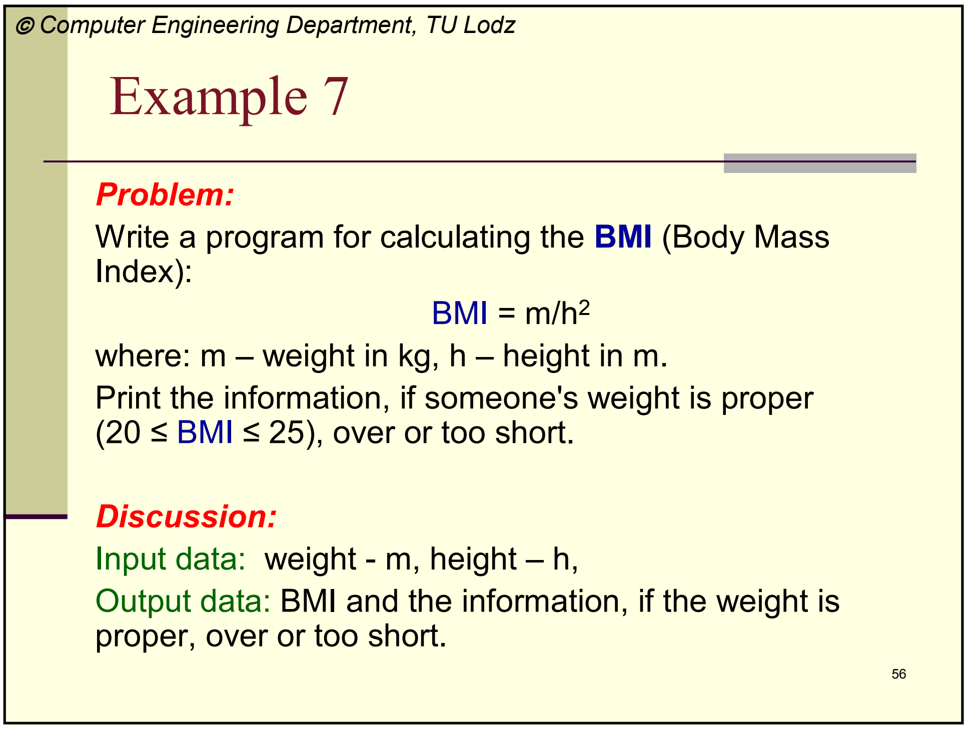
- Read ***n*** and check, if *n>0* .
- If "yes", read ***n*** numbers in a loop, and compute mean value.
- If "no" write "No numbers for calculations".



© Computer Engineering Department, TU Lodz



```
/*File mean_val.c         Program for calculating of arithmetic mean value */
#include <stdio.h>
int main()                              // main program
{
    int i, n;                           // loop index, number of data
    float x, sum;                       // input data, sum
    printf(" Program for calculating of mean value  \n");
    printf (" Enter the data number n = ");
    scanf("%d", &n);
    if(n>0)                                     // if statement, logical expression
    {                                   // begin of statement1
          sum=0.0;
          for(i=0; i<n; i++)            // begin of for loop
          {
              printf("Enter number %d: ", i+1);
              scanf("%f",&x);
              sum=sum+x;
          }                             // end of for loop
          printf("Mean value equals: %8.3f", sum/n);
    }
    else printf("No numbers for calculation\n");            // statement2
    return 0;
}
```



Example 7

Problem:

Write a program for calculating the **BMI** (Body Mass Index):

$$BMI = m/h^2$$

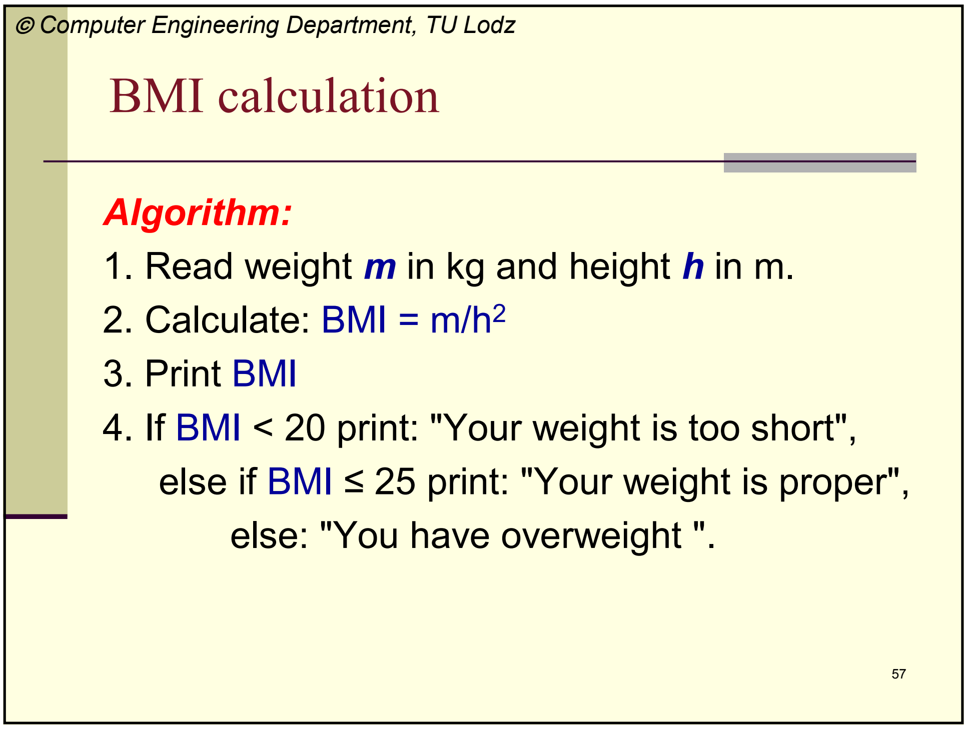
where: m – weight in kg, h – height in m.

Print the information, if someone's weight is proper (20 ≤ BMI ≤ 25), over or too short.

Discussion:

Input data: weight - m, height – h,

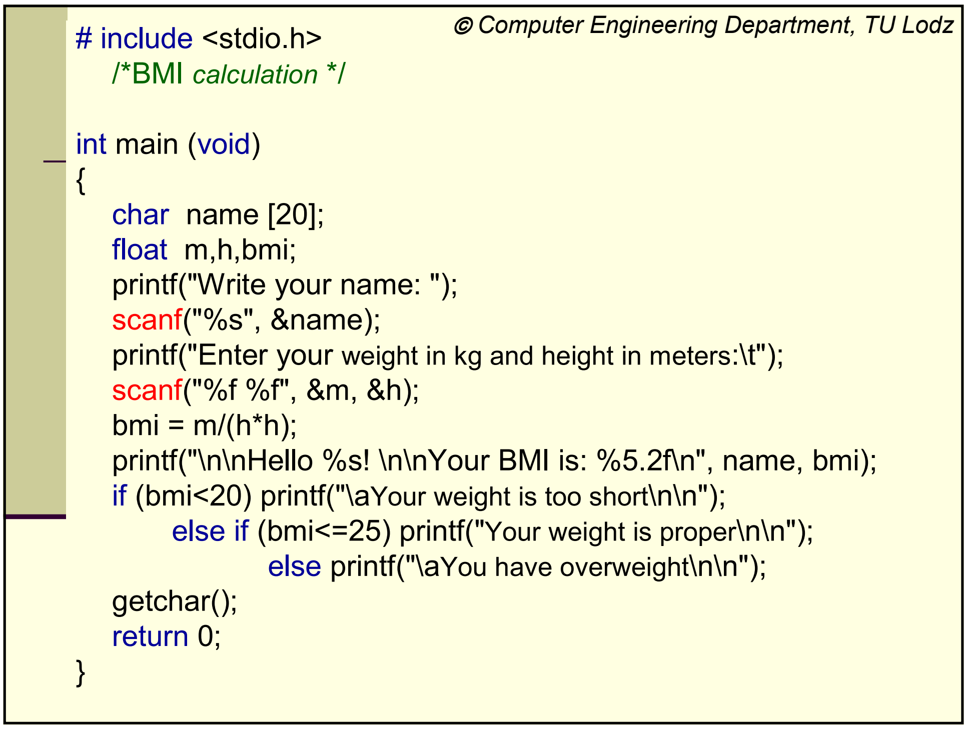
Output data: BMI and the information, if the weight is proper, over or too short.



BMI calculation

Algorithm:

1. Read weight ***m*** in kg and height ***h*** in m.
2. Calculate: BMI = m/h^2
3. Print BMI
4. If BMI < 20 print: "Your weight is too short",
   else if BMI ≤ 25 print: "Your weight is proper",
   else: "You have overweight ".



```
# include <stdio.h>
    /*BMI calculation */

int main (void)
{
    char  name [20];
    float  m,h,bmi;
    printf("Write your name: ");
    scanf("%s", &name);
    printf("Enter your weight in kg and height in meters:\t");
    scanf("%f %f", &m, &h);
    bmi = m/(h*h);
    printf("\n\nHello %s! \n\nYour BMI is: %5.2f\n", name, bmi);
    if (bmi<20) printf("\aYour weight is too short\n\n");
           else if (bmi<=25) printf("Your weight is proper\n\n");
                      else printf("\aYou have overweight\n\n");
    getchar();
    return 0;
}
```