Introduction to Programming in C

(IS-FEE-10061S)

Białystok University of Technology
Faculty of Electrical Engineering
Academic year 2023/2024

Workshop no. 02 (07.03.2024)

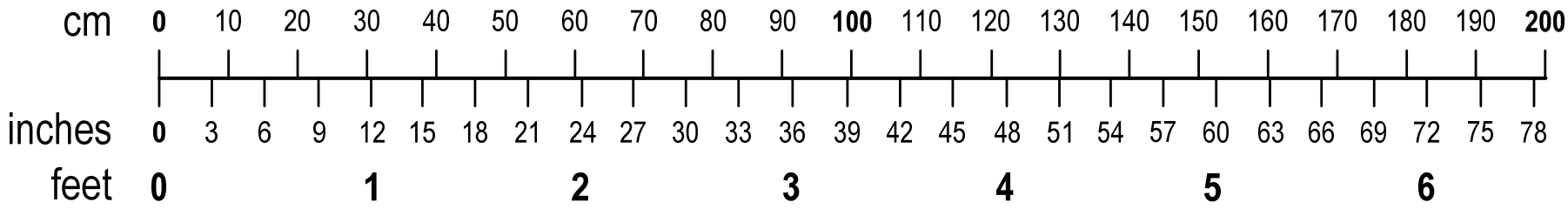
Jarosław Forenc, PhD

Topics

- Identifiers (names), keywords
- Data types
- Numerical constants, declarations of variables and constants
- Operators, expressions, statements
- Arithmetic expressions, mathematical functions (math.h)
- Functions: printf() and scanf()

Conversion of height in cm to feet and inches

- Units of length in the British system of measurement :
  - 1 inch [in] = 2.54 [cm]
  - 1 foot [ft] = 12 inches = 30.48 [cm]



Conversion of height in cm to feet and inches

```
Enter your height in cm: 175
175.000000 [cm] = 5.741470 [ft]
175.000000 [cm] = 68.897636 [in]
```

```c
#include <stdio.h>

int main(void)
{
    float cm;       /* height in cm */
    float feet;     /* height in feet */
    float inches;   /* height in inches */

    printf("Enter your height in cm: ");
    scanf("%f",&cm);

    feet = cm / 30.48;
    inches = cm / 2.54;

    printf("%f [cm] = %f [ft]\n",cm,feet);
    printf("%f [cm] = %f [in]\n",cm,inches);

    return 0;
}
```

Identifiers (names)

- Allowed characters: A-Z, a-z, 0-9, _ (underscore)
- Length is not limited (first 63 characters are distinguishable)
- Correct identifiers:

```
temp    u2    u_2    circle_area    alfa    Beta    XyZ
```

- The first character cannot be a number
- Spaces cannot be used in identifiers
- Incorrect identifiers:

```
2u    circle area
```

Identifiers (names)

- Identifiers should not be too long

```
temperature_in_celsius_scale
```

- The variable name should be related to its content
- The C language is case sensitive, so the following names mean different identifiers

```
tempc    Tempc    TempC    TEMPC    TeMpC
```

- C language keywords cannot be used as variable names

C language keywords

- There are 43 keywords defined in the C11 standard

`auto`	`extern`	`short`	`while`
`break`	`float`	`signed`	`_Alignas`
`case`	`for`	`sizeof`	`_Alignof`
`char`	`goto`	`static`	`_Bool`
`const`	`if`	`struct`	`_Complex`
`continue`	`inline`	`switch`	`_Generic`
`default`	`int`	`typedef`	`_Imaginary`
`do`	`long`	`union`	`_Noreturn`
`double`	`register`	`unsigned`	`_Static_assert`
`else`	`restrict`	`void`	`_Thread_local`
`enum`	`return`	`volatile`	

Data types

Name	Memory (bytes)	Range
`char`	1	-128 ... 127
`int`	4	-2147483648 ... 2147483647
`float`	4	$-3.4 \cdot 10^{38}$ ... $3.4 \cdot 10^{38}$
`double`	8	$-1.7 \cdot 10^{308}$ ... $1.7 \cdot 10^{308}$
`void`	–	–

- Keywords affecting types:
  - signed - signed number (char and int), e.g. signed char
  - unsigned - unsigned number (char and int), e.g. unsigned int
  - short, long, long long - short/long number (int), e.g. short int
  - long - greater precision (double), long double

Numerical constants (integers)

- Integer constants are written in the decimal system by default and have type int

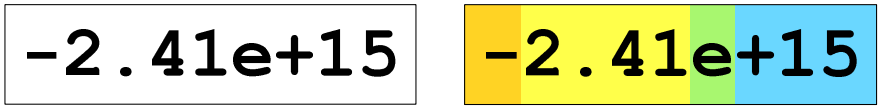
```
1     100     -125     123456
```

- Integers in other number systems
  - octal: 0 at the beginning, e.g. 011, 024
  - hexadecimal: 0x at the beginning, e.g. 0x2F, 0xab
- Suffixes at the end of the number change the type
  - l or L - long int type, e.g. 10l, 10L
  - ll or LL - long long int type, e.g. 10ll, 10LL
  - u or U - unsigned type, e.g. 10u, 10U

Numerical constants (floting-point)

- The default floating-point number type is double
- Format of writing floating-point constants

```
-2.41e+15
```



plus/minus sign	mantissa (digits with a decimal point)	e or E	exponent with a sign

- In the notation can be omitted:
  - plus sign, e.g. -2.41e15, 4.123E-3
  - decimal point or exponent, e.g. 2e-5, 14.15
  - fractional part or integer part, e.g. 2.e-5, .12e4
- Suffixes at the end of the number change the type:
  - l or L - long double type, e.g. 2.5l
  - f or F - float type, e.g. 3.14f

Declarations of variables and constants

- Variables - change their values while the program is running
- Constants - they have values set before the program starts and remain unchanged throughout the program's execution
- The declaration gives the variable/constant a name, specifies the type of value to be stored, and reserves the memory space accordingly
- Variable declarations:

```
int x;
float a, b;
char zn1;
```

- Constant declarations:

```
const int y = 5;
const float c = 1.25f;
const char zn2 = 'Q';
```

- Variable initialization:

```
int x = -10;
```

Symbolic constants (#define)

- The #define preprocessor directive allows you to define the so-called symbolic constants

#define constant_name constant_value

```
#define PI 3.14
#define MESSAGE "Start!!!\n"
```

- Symbolic constants are usually capitalized
- In the place of the constant, its value is inserted (before the compilation of the program)

Example: area and circumference of a circle

```c
#include <stdio.h>
#define PI 3.14
#define MESSAGE "Start!!!\n"

int main(void)
{
    double area, cf;
    double r = 1.5;

    printf(MESSAGE);
    area = PI * r * r;
    cf = 2 * PI * r;

    printf("Area = %g\n", area);
    printf("Circumference = %g\n", cf);

    return 0;
}
```

Example: area and circumference of a circle

```
/***
...
#endif  /* _INC_STDIO */

int main(void)
{
    double area, cf;
    double r = 1.5;

    printf("Start!!!\n");
    area = 3.14 * r * r;
    cf = 2 * 3.14 * r;

    printf("Area = %g\n", area);
    printf("Circumference = %g\n", cf);

    return 0;
}
```

stdio.h file content

```
Start!!!
Area = 7.065
Circumference = 9.42
```

Operators

Type	Symbol
Arithmetic	`+ - * / %`
Increment / decrement	`++ --`
Relational	`< > <= >= == !=`
Logical	`&& || !`
Bitwise	`& | ^ << >> ~`

Assignment	`=`
	`+= -= *= /= %=`
	`<<= >>= &= |= ^=`

Other	`() [] & * -> .` `, ? : sizeof (type)`

Expressions and statements

- Expression - combination of operators and operands

```
4     -6     4+2.1     x=5+2     a>3     x>5&&x<8
```

- Each expression has a type and a value
- Statement - the main element of which the program is built, ends with a semicolon

Expression:

```
x = 5
```

Statement:

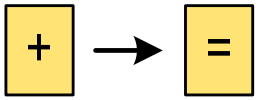
```
x = 5;
```

- The C language considers any expression that ends with a semicolon to be a statement

Arithmetic expressions

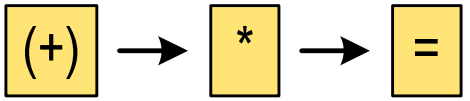
- Arithmetic expressions can consist of:
  - numerical constants, variables, constants
  - operators: `+ - * / % = ( )` and others
  - function calls (math.h file header)
- The order in which operations are performed depends on the priority of the operators

```
w = a + b;
```

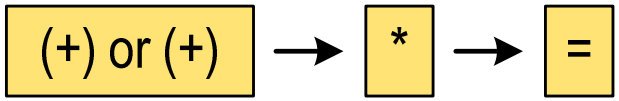



```
w = a + b * c;
```


```
w = (a + b) * c;
```




```
w = (a + b) * (c + d);
```



Arithmetic expressions

- Operation order

```
w = a + b + c;        →   w = ((a + b) + c);
w = x = y = a + b;    →   w = (x = (y = (a + b)));
```

- Division in arithmetic expressions

$$w = \frac{a+b}{c+d}$$

```
w = a + b / c + d;
```
INCORRECTLY

```
w = (a + b) / (c + d);
```
CORRECTLY

$$w = \frac{a+b}{c \cdot d}$$

```
w = (a + b) / c * d;
```
INCORRECTLY

```
w = (a + b) / (c * d);
```
CORRECTLY

Arithmetic expressions

- When dividing integers, the fractional part is discarded

$$w = \frac{5}{4}$$

```
5 / 4 = 1
```

```
5.0 / 4 = 1.25
```

```
5 / 4.0 = 1.25
```

```
5.0 / 4.0 = 1.25
```

```
5.0f / 4 = 1.25
```

```
5. / 4 = 1.25
```

```
(float) 5 / 4 = 1.25
```

Casting: (type)

Mathematical functions (math.h)

- The header file math.h contains definitions of selected constants

Name	Value	Description
`M_PI`	3.14159265358979323846	π number
`M_E`	2.71828182845904523536	*e* - Euler's number
`M_LN2`	0.69314718055994530941 7	ln 2
`M_SQRT2`	1.41421356237309504880	$\sqrt{2}$

Mathematical functions (math.h)

- Sample math functions:

Name	Declaration	Description
`abs`	**int abs(int x);**	absolute value of x (x - integer)
`fabs`	**double fabs(double x);**	absolute value of x (x - float-pointing)
`sqrt`	**double sqrt(double x);**	square root of x
`pow`	**double pow(double x, double y);**	x^y - x to the y power
`sin`	**double sin(double x);**	sine of x in radians
`atan`	**double atan(double x);**	arc tangent of x
`atan2`	**double atan2(double y, double x);**	arc tangent of y/x quotient

- All functions have three versions - for float, double and long double arguments

Example: resonant frequency

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

int main(void)
{
    double L, C, fr;

    printf("Enter L [H]:  "); scanf("%lf",&L);
    printf("Enter C [F]:  "); scanf("%lf",&C);

    fr = 1/(2*M_PI*sqrt(L*C));

    printf("--------------------\n");
    printf("fr [Hz]:  %.3f\n",fr);

    return 0;
}
```

```
Enter L [H]:  0.01
Enter C [F]:  1e-6
--------------------
fr [Hz]:  1591.549
```

$$f_r = \frac{1}{2\pi\sqrt{LC}}$$

printf() function

- General syntax for the printf() function

```
printf("control_statement",arg1,arg2,...);
```

- In its simplest form, printf() displays only text

```
printf("Hello world");
```

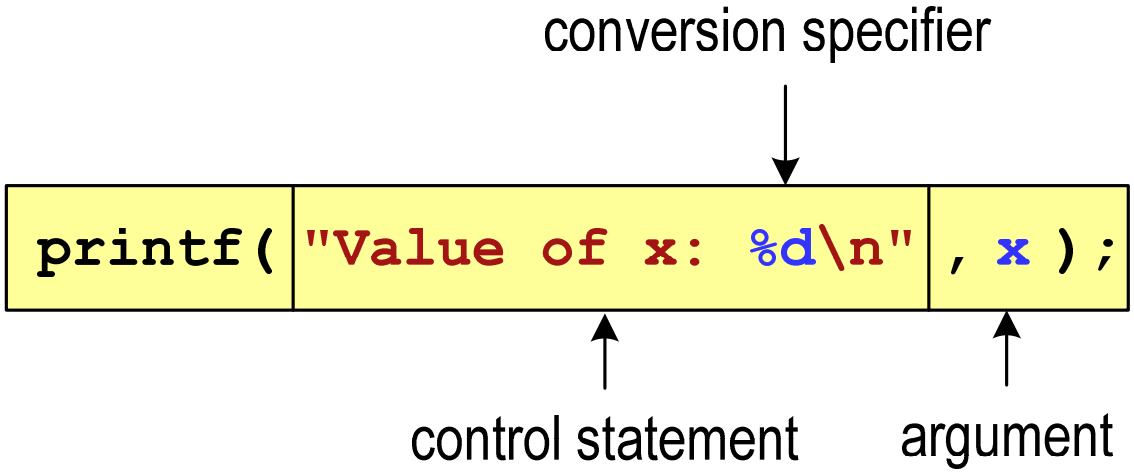
```
Hello world
```

- To display variable values, it is necessary to use conversion specifiers that specify the type and display of the arguments

```
%[flag][width][.precision][modifier]type
```

printf() function

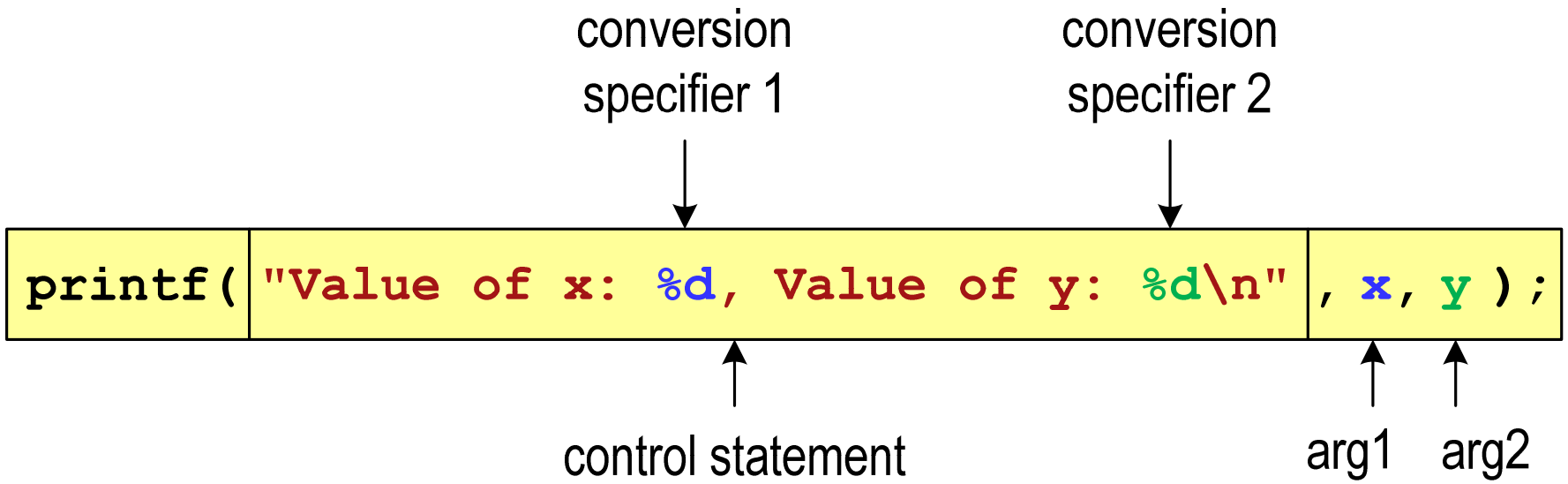
```
int x = 10;
printf("Value of x: %d\n", x);
```




```
Value of x: 10
```

printf() function

```
int x = 10, y = 20;
printf("Value of x: %d, Value of y: %d\n", x, y);
```




```
Value of x: 10, Value of y: 20
```

Conversion specifiers (printf)

Type in C	Specifier	Meaning
`char`	`%c`	single character
	`%d`	character ASCII code, decimal integer
`char *`	`%s`	character string
`int`	`%d  %i`	signed decimal integer
	`%o %O`	unsigned octal integer
	`%x  %X`	unsigned hexadecimal integer
`float` `double`	`%f`	floating-point number, decimal notation
	`%e  %E`	floating-point number, e-notation
	`%g  %G`	floating-point number (%f or %e)

printf() function

```
int x = 123; float y = 1.23456789f;
```

```
printf("x = [%d], y = [%f]\n", x, y);
```

```
x = [123], y = [1.234568]
```

```
printf("x = [], y = []\n", x, y);
```

```
x = [], y = []
```

```
printf("x = [%d], y = [%d]\n", x, y);
```

```
x = [123], y = [-536870912]
```

printf() function

```
int x = 123; float y = 1.23456789f;
```

```
printf("x = [%6d], y = [%12f]\n", x, y);
```

```
x = [   123], y = [    1.234568]
```

```
printf("x = [%6d], y = [%12.3f]\n", x, y);
```

```
x = [   123], y = [       1.235]
```

```
printf("x = [%6d], y = [%.3f]\n", x, y);
```

```
x = [   123], y = [1.235]
```

scanf() function

- General syntax for the scanf() function

```
scanf("specifiers",argument_adresses);
```

- Conversion specifier syntax

```
%[width][modifier]type
```

- The arguments are addresses of memory areas, so they must be preceded by the & sign

```
int x;
scanf("%d", &x);
```

scanf() function

- The conversion specifiers are in most cases the same as for the printf() function
- The difference is between the float and double types

Type in C	Specifier	Meaning
`float`	`%f`	floating-point number
	`%e` `%E`	floating-point number, e-notation
	`%g` `%G`	floating-point number (%f or %e)
`double`	`%lf`	floating-point number
	`%le` `%lE`	floating-point number, e-notation
	`%lg` `%lG`	floating-point number (%f or %e)

scanf() function

```
int a, b, c;
scanf("%d %d %d", &a, &b, &c);
```

- The arguments can be separated from each other by any number of white (non-printing) characters: space, tab, enter

```
15 20 -30
```

```
15 20 -30<enter>
```

```
   15   20    -30
```

```
   15   20    -30<enter>
```

```
15
20
-30
```

```
15<enter>
20<enter>
-30<enter>
```

End of workshop no. 02

Thank you for your attention!