EECS 10: Computational Methods in Electrical and Computer Engineering

Lecture 2

Rainer Dömer

doemer@uci.edu

The Henry Samueli School of Engineering
Electrical Engineering and Computer Science
University of California, Irvine

Lecture 2.1: Overview

- Review Quiz
- Our second C Program
  - Program structure
    - Input
    - Computation
    - Output
  - Example **`Addition.c`**
    - Variables
    - Value input
    - Calculation
    - Result output

Quiz: Question 1

- Which Linux command shows you the path to the current directory?
  a) **`cd`**
  b) **`pwd`**
  c) **`dir`**
  d) **`ls`**
  e) **`list`**

Quiz: Question 1

- Which Linux command shows you the path to the current directory?
  a) **`cd`**
  b) **`pwd`**
  c) **`dir`**
  d) **`ls`**
  e) **`list`**

Quiz: Question 2

- Which of the following Linux commands renames file “text1” into “homework1”?
  a) **`ren text1 homework1`**
  b) **`ren homework1 text1`**
  c) **`rm text1 homework1`**
  d) **`mv homework1 text1`**
  e) **`mv text1 homework1`**

Quiz: Question 2

- Which of the following Linux commands renames file “text1” into “homework1”?
  a) **`ren text1 homework1`**
  b) **`ren homework1 text1`**
  c) **`rm text1 homework1`**
  d) **`mv homework1 text1`**
  e) **`mv text1 homework1`**

Quiz: Question 3

- What is C *not*?
  a) a structured programming language
  b) a object-oriented programming language
  c) a compiled programming language
  d) a high-level programming language
  e) a portable programming language

Quiz: Question 3

- What is C *not*?
  a) a structured programming language
  b) a object-oriented programming language
  c) a compiled programming language
  d) a high-level programming language
  e) a portable programming language

Quiz: Question 4

- What is the meaning of the following code fragment?

```
/* printf("C programming is great!\n") */
```

a) it prints “C programming is boring!”
b) it prints “C programming is great!”
c) it is a syntax error because a semicolon is missing after the **`printf()`** statement
d) it is the main function of the C program
e) it is a comment ignored by the compiler

Quiz: Question 4

- What is the meaning of the following code fragment?

```
/* printf("C programming is great!\n") */
```

a) it prints “C programming is boring!”
b) it prints “C programming is great!”
c) it is a syntax error because a semicolon is missing after the **`printf()`** statement
d) it is the main function of the C program
e) it is a comment ignored by the compiler

Quiz: Question 5

- What is true about of the following compiler call? (Check all that apply!)

```
% gcc HelloWorld.c –Wall –ansi –o HelloWorld
```

a) the GNU C Compiler is called to generate an executable program called **`HelloWorld`**

b) the compiler will print warning and/or error messages about any non-ANSI compliance in the code

c) the compiler will ignore all warnings

d) the compiler will read the file **`HelloWorld.c`**

e) the compiler will overwrite the **`HelloWorld`** file if it already exists

Quiz: Question 5

- What is true about of the following compiler call? (Check all that apply!)

```
% gcc HelloWorld.c –Wall –ansi –o HelloWorld
```

a) the GNU C Compiler is called to generate an executable program called **`HelloWorld`**

b) the compiler will print warning and/or error messages about any non-ANSI compliance in the code

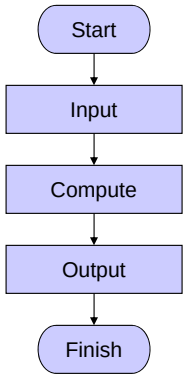
c) the compiler will ignore all warnings

d) the compiler will read the file **`HelloWorld.c`**

e) the compiler will overwrite the **`HelloWorld`** file if it already exists

Program Structure

- General Program Structure
  - Input
    - read input data
  - Computation
    - compute output data from input data
  - Output
    - write output data
- Examples
  - Calculator
    - Enter numbers, compute function, output result
  - Word processor
    - Type, format, print text
  - Database application
    - Enter data, process data, present data
  - etc.

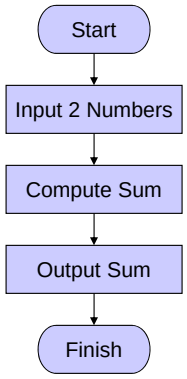


C Program Structure

- Initialization section
  - Definition of variables (storage elements)
    - Name, type, and initial value
- Input section
  - read values from input devices into variables
    - standard input functions
- Computation section
  - perform the necessary computation on variables
    - assignment statements
- Output section
  - write results from variables to output devices
    - standard output functions
- Exit section
  - clean up and exit

Our second C Program

- Program Example: Addition
  - Input
    - Let the user enter two whole numbers
  - Computation
    - Compute the sum of the two numbers
  - Output
    - Display the sum



Our second C Program

- Program example: **`Addition.c`** (part 1/2)

```
/* Addition.c: adding two integer numbers       */
/*                                              */
/* author: Rainer Doemer                        */
/*                                              */
/* modifications:                               */
/* 09/30/04 RD  initial version                 */

#include <stdio.h>

/* main function */

int main(void)
{
    /* variable definitions */
    int i1 = 0;       /* first integer */
    int i2 = 0;       /* second integer */
    int sum;          /* result */
...
```

Our second C Program

- Program example: **`Addition.c`** (part 2/2)

```
...
    /* input section */
    printf("Please enter an integer:      ");
    scanf("%d", &i1);
    printf("Please enter another integer: ");
    scanf("%d", &i2);

    /* computation section */
    sum = i1 + i2;

    /* output section */
    printf("The sum of %d and %d is %d.\n", i1, i2, sum);

    /* exit */
    return 0;
} /* end of main */

/* EOF */
```


Our second C Program

- Variable definition and initialization

```
    /* variable definitions */
    int i1 = 0;       /* first integer */
    int i2 = 0;       /* second integer */
    int sum;          /* result */
```

  - Variable type: **`int`**
    - integer type, stores whole numbers (e.g. -5, 0, 42)
    - many other types exist (**`float`**, **`double`**, **`char`**, ...)
  - Variable name: **`i1`**
    - valid identifier, i.e. name composed of letters, digits
    - variable name should be descriptive
  - Initializer: **`= 0`**
    - specifies the initial value of the variable
    - optional (if omitted, initial value is undefined)

Our second C Program

- Data input using **`scanf()`** function

```
/* input section */
printf("Please enter an integer:      ");
scanf("%d", &i1);
```

  - Function **`scanf()`** is defined in standard I/O library
    - declared in header file **`stdio.h`**
  - … reads data from the standard input stream **`stdin`**
    - **`stdin`** usually means the keyboard
  - … converts input data according to format string
    - **`"%d"`** indicates that a decimal integer value is expected
  - … stores result in specified location
    - **`&i1`** indicates to store at the *address of* variable **`i1`**

Our second C Program

- Computation using assignment statements

```
/* computation section */
sum = i1 + i2;
```

  - Operator **`=`** specifies an assignment
    - value of the right-hand side (**`i1 + i2`**) is assigned to the left-hand side (**`sum`**)
    - left-hand side is usually a variable
    - right-hand side is a simple or complex expression
  - Operator **`+`** specifies addition
    - left and right arguments are added
    - result is the sum of the two arguments
  - Many other operators exist
    - For example, **`-`**, **`*`**, **`/`**, **`%`**, **`<`**, **`>`**, **`==`**, **`^`**, **`&`**, **`|`**, ...

Our second C Program

- Data output using **`printf()`** function

```
/* output section */
printf("The sum of %d and %d is %d.\n", i1, i2, sum);
```

  - Function **`printf()`** is defined in standard I/O library
    - declared in header file **`stdio.h`**
  - … writes data to the standard output stream **`stdout`**
    - **`stdout`** usually means the monitor
  - … converts output data according to format string
    - text (**"The sum…"**) is copied verbatim to the output
    - **"%d"** is replaced with a decimal integer value
  - … takes values from specified arguments (in order)
    - **`i1`** indicates to use the value of the variable **`i1`**

Our second C Program

- Example session: **`Addition.c`**

```
% vi Addition.c
% ls -l
-rw-------   1 doemer   faculty      702 Sep 30 14:17 Addition.c
% gcc -Wall -ansi Addition.c -o Addition
% ls -l
-rwx------   1 doemer   faculty     6628 Sep 30 16:44 Addition*
-rw-------   1 doemer   faculty      702 Sep 30 14:17 Addition.c
% Addition
Please enter an integer:      27
Please enter another integer: 15
The sum of 27 and 15 is 42.
% Addition
Please enter an integer:      123
Please enter another integer: -456
The sum of 123 and -456 is -333.
%
```

Lecture 2.2: Overview

- Basic Types in C
  - Integer types
  - Floating point types
- Arithmetic Operations in C
  - Arithmetic operators
  - Evaluation order
- Arithmetic Example
  - Cosine approximation
  - Example **`Cosine.c`**

Basic Types in C

- Integer types
  - **`char`** Character, e.g. **`'a'`**, **`'b'`**, **`'1'`**, **`'*'`**
    - typical range **`[-128,127]`**
  - **`short int`** Short integer, e.g. **`-7`**, **`0`**, **`42`**
    - typical range **`[-32768,32767]`**
  - **`int`** Integer, e.g. **`-7`**, **`0`**, **`42`**
    - typical range **`[-2147483648,2147483647]`**
  - **`long int`** Long integer, e.g. **`-99L`**, **`9L`**, **`123L`**
    - typical range **`[-2147483648,2147483647]`**
  - **`long long int`** Very long integer, e.g. **`12345LL`**
    - typical range **`[-9223372036854775808,9223372036854775807]`**
- Integer types can be
  - **`signed`** negative and positive values (incl. 0)
  - **`unsigned`** positive values only (incl. 0)

Basic Types in C

- Floating point types
  - **`float`** Floating point with single precision
    - Example **`3.5f`**, **`-0.234f`**, **`10e8f`**
  - **`double`** Floating point with double precision
    - Example **`3.5`**, **`-0.23456789012`**, **`10e88`**
  - **`long double`** Floating point with high precision
    - Example **`12345678.123456e123L`**
- Floating point values are in many cases *approximations* only!
  - Storage size of floating point values is fixed
  - Many values can only be represented as approximations
  - Example: **`1.0/3.0 = .333333`**

EECS10: Computational Methods in ECE, Lecture 2 (c) 2013 R. Doemer 25

Conversion Specifiers for Basic Types

Type	`printf()`	`scanf()`
`long double`	**`%Lf`**	**`%Lf`**
`double`	**`%f`**	**`%lf`**
`float`	**`%f`**	**`%f`**
`unsigned long long`	**`%llu`**	**`%llu`**
`long long`	**`%lld`**	**`%lld`**
`unsigned long`	**`%lu`**	**`%lu`**
`long`	**`%ld`**	**`%ld`**
`unsigned int`	**`%u`**	**`%u`**
`int`	**`%d`**	**`%d`**
`short`	**`%hd`**	**`%hd`**
`char`	**`%c`**	**`%c`**

EECS10: Computational Methods in ECE, Lecture 2 (c) 2013 R. Doemer 26

Arithmetic Operations in C

- Arithmetic Operators
  - parentheses **(,)**
  - unary plus, minus **+, -**
  - multiplication, division, modulo ***, /, %**
  - addition, subtraction **+, -**
  - shift left, shift right **<<, >>**
- Evaluation order of expressions
  - usually left to right
  - by operator precedence
    - ordered as in table above (higher operators are evaluated first)
- Arithmetic operators are available
  - for integer types: all
  - for floating point types: all except **%, <<, >>**

Shift Operators

- Left-shift operator: **x << n**
  - shifts *x* in binary representation *n* times to the left
  - multiplies *x* *n* times by 2
  - Examples
    - $2x$ = **x << 1**
    - $4x$ = **x << 2**
    - $x * 2^n$ = **x << n**
    - 2^n = **1 << n**
- Right-shift operator: **x >> n**
  - shifts *x* in binary representation *n* times to the right
  - divides *x* *n* times by 2
  - Examples
    - $x / 2$ = **x >> 1**
    - $x / 4$ = **x >> 2**
    - $x / 2^n$ = **x >> n**

Example Program

- Cosine function approximation
  - Task
    - Design a program to compute the cosine function!
    - In your program, use only the four basic operations addition, subtraction, multiplication, and division.
  - Approach
    - The cosine function can be algebraically approximated using an infinite sum

$$\cos x = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n}}{(2n)!} \approx 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \ldots$$

Example Program

- Program example: **`Cosine.c`** (part 1/2)

```
/* Cosine.c: cosine function approximation      */
/*                                              */
/* author: Rainer Doemer                        */
/*                                              */
/* modifications:                               */
/* 10/02/05 RD  initial version                 */

#include <stdio.h>

/* main function */

int main(void)
{
    /* variable definitions */
    double x, y;

    /* input section */
    printf("Please enter real value x: ");
    scanf("%lf", &x);
...
```

Example Program

- Program example: **`Cosine.c`** (part 2/2)

```
...

    /* computation section */
    y = 1 - (x*x)/(2.0*1.0)
          + (x*x*x*x)/(4.0*3.0*2.0*1.0)
          - (x*x*x*x*x*x)/(6.0*5.0*4.0*3.0*2.0*1.0);

    /* output section */
    printf("cos(%f) is approximately %f\n", x, y);

    /* exit */
    return 0;
} /* end of main */

/* EOF */
```


Example Program

- Example session: **`Cosine.c`**

```
% vi Cosine.c
% gcc -Wall -ansi Cosine.c -o Cosine
% Cosine
Please enter real value x: 0.0
cos(0.000000) is approximately 1.000000
% Cosine
Please enter real value x: 0.1
cos(0.100000) is approximately 0.995004
% Cosine
Please enter real value x: 1.57079
cos(1.570790) is approximately -0.000888
% Cosine
Please enter real value x: 3.1415927
cos(3.141593) is approximately -1.211353
%
```

Lecture 2.3: Overview

- Review Quiz
- Type Conversion
  - explicit
  - implicit
- Types in Expressions
- Arithmetic Computation
  - Example **`Arithmetic.c`**

Quiz: Question 6

- Which of the following constructs is a valid arithmetic operator in C?
  (Check all that apply!)
  a) `/`
  b) `%`
  c) `!`
  d) `@`
  e) `>>`

Quiz: Question 6

- Which of the following constructs is a valid arithmetic operator in C?
  (Check all that apply!)

a) **/**
b) **%**
c) **!**
d) **@**
e) **>>**

Quiz: Question 7

- What is the value of the integer **x** after the following statement?

```
x = 11 / 3 + 11 % 3;
```

a) **1**
b) **2**
c) **3**
d) **4**
e) **5**

Quiz: Question 7

- What is the value of the integer **x** after the following statement?

```
x = 11 / 3 + 11 % 3;
```

a) **1**
b) **2**
c) **3**
d) **4**
e) **5**

Quiz: Question 8

- What is the value of the integer **x** after the following statement?

```
x = (10 – (3 – (20 - -10)));
```

a) **7**
b) **17**
c) **27**
d) **37**
e) **77**

Quiz: Question 8

- What is the value of the integer **x** after the following statement?

```
x = (10 – (3 – (20 - -10)));
```

a) **7**
b) **17**
c) **27**
d) **37**
e) **77**

Quiz: Question 9

- Which of the following format strings will print an **unsigned int** value in decimal format when used with **printf()**?

a) **"%u"**
b) **"%ud"**
c) **"%d"**
d) **"%lu"**
e) **"%ui"**

Quiz: Question 9

- Which of the following format strings will print an **`unsigned int`** value in decimal format when used with **`printf()`**?

a) **`"%u"`**
b) `"%ud"`
c) `"%d"`
d) `"%lu"`
e) `"%ui"`

Quiz: Question 10

- Which of the following statements will correctly read a decimal value from **`stdin`** into a variable **`x`** of type **`signed int`**?

a) **`stdin("%x", &u);`**
b) **`stdin("%u", x);`**
c) **`scanf("%d", &x);`**
d) **`scanf("&x", %u);`**
e) **`scanf("&x", %d);`**

Quiz: Question 10

- Which of the following statements will correctly read a decimal value from **`stdin`** into a variable **`x`** of type **`signed int`**?

a) `stdin("%x", &u);`
b) `stdin("%u", x);`
c) **`scanf("%d", &x);`**
d) `scanf("&x", %u);`
e) `scanf("&x", %d);`

Review: Basic Types in C

- Integer types
  - **`char`** Character, e.g. **`'a'`**, **`'b'`**, **`'1'`**, **`'*'`**
    - typical range **`[-128,127]`**
  - **`short int`** Short integer, e.g. **`-7`**, **`0`**, **`42`**
    - typical range **`[-32768,32767]`**
  - **`int`** Integer, e.g. **`-7`**, **`0`**, **`42`**
    - typical range **`[-2147483648,2147483647]`**
  - **`long int`** Long integer, e.g. **`-99L`**, **`9L`**, **`123L`**
    - typical range **`[-2147483648,2147483647]`**
  - **`long long int`** Very long integer, e.g. **`12345LL`**
    - typical range **`[-9223372036854775808,9223372036854775807]`**
- Integer types can be
  - **`signed`** negative and positive values (incl. 0)
  - **`unsigned`** positive values only (incl. 0)

Review: Basic Types in C

- Floating point types
  - **`float`** Floating point with single precision
    - Example **`3.5f`**, **`-0.234f`**, **`10e8f`**
  - **`double`** Floating point with double precision
    - Example **`3.5`**, **`-0.23456789012`**, **`10e88`**
  - **`long double`** Floating point with high precision
    - Example **`12345678.123456e123L`**
- Floating point values are in many cases *approximations* only!
  - Storage size of floating point values is fixed
  - Many values can only be represented as approximations
  - Example: **`1.0/3.0 = .333333`**

Type Conversion

- Explicit Type Conversion
  - types can be explicitly converted to other types, by use of the type cast operator:
    `(type) expression`
  - the target type is named explicitly in parentheses before the source expression
  - Examples:
    - **`Float = (float) LongInt`**
      - converts the **`long int`** value into a **`float`** value
    - **`Integer = (int) Double`**
      - converts the **`double`** value into an **`int`** value
      - any fractional part is truncated!
    - **`Char = (char) LongLongInt`**
      - converts the **`long long int`** value into a **`char`** value
      - any out-of-range values are silently cut off!

Type Conversion

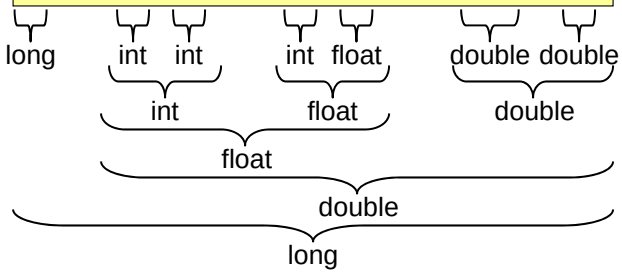
- Implicit Type Conversion
  - Type promotion
    - integral promotion
      - **unsigned** or **signed char** is promoted to **unsigned** or **signed int** before any operation
      - **unsigned** or **signed short** is promoted to **unsigned** or **signed int** before any operation
    - binary arithmetic operators are defined only for same types
      - the smaller type is converted to the larger type (before operation)
      - Examples:
        - » **ShortInt * LongInt** results in a **long int** type
        - » **LongDouble * Float** results in a **long double** type
  - Type coercion
    - most types are automatically converted to expected types
    - Example: **Double = Float**, or **Char = LongInt**

Types in Expressions

- Expressions are composed of constants, variables and operators, each of which has an associated type
- Example:

```
short int s;
int       i;
long int  l;
float     f;
double    d;

l  =   2 * s   +   i * f   -   0.5  * d;
```



Example Program

- Program example:
  - Task: Write a C program that exercises arithmetic computation by use of different types and operators!
  - The program should compute the following equations:
    - Polynomial:

$$p = 2x^2 - 3x + 5$$

    - Quotient of sums:

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

    - Remainder:

$$r = rem(2^n / 7)$$

  - Assume that a, b, c, d, and n are whole numbers.

Example Program

- Program example: **`Arithmetic.c`** (part 1/3)

```
/* Arithmetic.c: arithmetic expresions          */
/*                                              */
/* author: Rainer Doemer                        */
/*                                              */
/* modifications:                               */
/* 10/06/04 RD  initial version                 */

#include <stdio.h>

/* main function */

int main(void)
{
    /* variable definitions */
    int    a, b, c, d, n;
    double p, q, r, x;

...
```

Example Program

- Program example: **`Arithmetic.c`** (part 2/3)

```
...

    /* input section */
    printf("Please enter the value for real x:    ");
    scanf("%lf", &x);
    printf("Please enter the value for integer a: ");
    scanf("%d", &a);
    printf("Please enter the value for integer b: ");
    scanf("%d", &b);
    printf("Please enter the value for integer c: ");
    scanf("%d", &c);
    printf("Please enter the value for integer d: ");
    scanf("%d", &d);
    printf("Please enter the value for integer n: ");
    scanf("%d", &n);

...
```


Example Program

- Program example: **`Arithmetic.c`** (part 3/3)

```
...

    /* computation section */
    p = 2.0*x*x - 3.0*x + 5.0;
    q = ((double)(a + b)) / ((double)(c + d));
    r = (1<<n) % 7;

    /* output section */
    printf("The value for the polynomial p is %f.\n", p);
    printf("The value for the quotient   q is %f.\n", q);
    printf("The value for the remainder  r is %f.\n", r);

    /* exit */
    return 0;
} /* end of main */

/* EOF */
```

Example Program

- Example session: **`Arithmetic.c`**

```
% vi Arithmetic.c
% gcc Arithmetic.c -Wall -ansi -o Arithmetic
% ls -l
total 20
-rwx------   1 doemer   faculty     7344 Oct  6 08:42 Arithmetic*
-rw-------   1 doemer   faculty     1154 Oct  6 08:37 Arithmetic.c
% Arithmetic
Please enter the value for real x:    3.1415927
Please enter the value for integer a: 5
Please enter the value for integer b: 6
Please enter the value for integer c: 7
Please enter the value for integer d: 8
Please enter the value for integer n: 9
The value for the polynomial p is 15.314431.
The value for the quotient   q is 0.733333.
The value for the remainder  r is 1.000000.
%
```