

EECS 10: Computational Methods in Electrical and Computer Engineering

Review of Lectures 1 - 8

Rainer Dömer

doemer@uci.edu

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

Review of Lectures 1 - 8

- Lecture 1: Course administration, setup
- Lecture 2: Unix system environment
- Lecture 3: Introduction to C programming
- Lecture 4: Input, computation, output
- Lecture 5: Basic types, operators
- Lecture 6: Arithmetic expressions
- Lecture 7: Conditional operators, statements
- Lecture 8: Counters, repetition statements

Introduction

- Course Contents
 - Introduction to computers
 - Introduction to structured programming
 - C, a high-level structured programming language
 - Binary data representation
 - Introduction to algorithm efficiency
 - Solving engineering problems
 - Applications of structured programming
 - Hands-on experience
 - Laboratory and discussion sessions

Course Administration

- Course web pages online at <http://eee.uci.edu/08f/18010/>
 - Instructor information
 - Course description and contents
 - Course policies and resources
 - Course schedule
 - Homework assignments
 - Course communication
 - Noteboard (announcements and technical discussion)
 - Email (administrative issues)

Getting Started

- Obtain your UCI netID
 - Your unique ID at UCI
 - Activation online at NACS web pages:

`http://activate.uci.edu/activate/menu.html`
- Obtain an account on the EECS servers
 - Your working account in EECS
 - Activation online at EECS web pages:

`https://newport.eecs.uci.edu/account.py`

Getting Started

- Log into the server
 - Use a terminal with SSH protocol (secure shell)
 - Connect to an EECS server
 - `malibu.eecs.uci.edu`
 - `vivian.eecs.uci.edu`
 - `newport.eecs.uci.edu`
 - Authorize yourself with user name and password
- Work in the Unix system environment
 - Unix shell prints command prompt, awaiting input
 - Type in system commands
`echo, date, ls, cat, man, more,`
`pwd, mkdir, cd, cp, mv, rm, rmdir`
 - Refer to manual pages for help on commands

Introduction to Computers

- What is a computer?
 - Digital device capable of executing programs
 - performing computations
 - making logical decisions
- What is a program?
 - Set of instructions which process data
 - input data (e.g. from keyboard, mouse, disk)
 - output data (e.g. to monitor, printer, disk)
- What is programming?
 - Creation of computer programs by use of a programming language

EECS10: Computational Methods in ECE, Review 1-8

(c) 2008 R. Doemer

7

Unix System Environment

- Unix system commands
 - **echo** print a message
 - **date** print the current date and time
 - **ls** list the contents of the current directory
 - **cat** list the contents of files
 - **more** list the contents of files page by page
 - **pwd** print the path to the current working directory
 - **mkdir** create a new directory
 - **cd** change the current directory
 - **cp** copy a file
 - **mv** rename and/or move a file
 - **rm** remove (delete) a file
 - **rmdir** remove (delete) a directory
 - **man** view manual pages for system commands

EECS10: Computational Methods in ECE, Review 1-8

(c) 2008 R. Doemer

8

Unix System Environment

- Text editing
 - **vi** standard Unix editor
 - **vim** vi-improved (supports syntax highlighting)
 - **pico** easy-to-use text editor
 - **emacs** very powerful editor
 - many others...
- Pick one editor and make yourself comfortable with it!

Unix System Environment

- Example session (1/4):

```
login as: doemer
Password:
Last login: Mon Oct  1 08:20:09 2007 from beta.eecs.uci.e
...
If this system is busy, consider a less loaded one below:
vivian.eecs.uci  up 30 days, 18:00,  load average: 0.00, 0.00, 0.01
malibu.eecs.uci up 2826 days, 21:06, load average: 0.00, 0.00, 0.01
newport.eecs.uc up 23 days, 23:29,  load average: 0.00, 0.00, 0.02
east.eecs.uci.e up 12 days,  4:56,  load average: 1.46, 1.41, 1.68
doemer@vivian% date
Mon Oct  1 08:24:47 PDT 2007
doemer@vivian% echo "Hello EECS10!"
Hello EECS10!
doemer@vivian% ls
eecs10/          Mail/            tmp/
doemer@vivian% pwd
/users/faculty/doemer
doemer@vivian% mkdir homework
doemer@vivian% ls
eecs10/          homework/       Mail/           tmp/
...
```

Unix System Environment

- Example session (2/4):

```

...
doemer@vivian% cd homework
doemer@vivian% pwd
/users/faculty/doemer/homework
doemer@vivian% ls
doemer@vivian% mkdir hw1
doemer@vivian% ls
hw1/
doemer@vivian% cd hw1
doemer@vivian% ls
doemer@vivian% vi program.c
doemer@vivian% ls
program.c
doemer@vivian% ls -l
total 2
-rw----- 1 doemer smmsp      51 Oct  1 08:32 program.c
doemer@vivian% more program.c
This is my new program file.
I don't know C yet...
...

```

Unix System Environment

- Example session (3/4):

```

...
doemer@vivian% cp program.c mybackup.c
doemer@vivian% ls
mybackup.c  program.c
doemer@vivian% ls -l
-rw----- 1 doemer smmsp      51 Oct  1 08:34 mybackup.c
-rw----- 1 doemer smmsp      51 Oct  1 08:32 program.c
doemer@vivian% cd ..
doemer@vivian% pwd
/users/faculty/doemer/homework
doemer@vivian% ls
hw1/
doemer@vivian% /ecelib/bin/turnin
=====
EECS 10 Fall 2007:
Assignment "hw1" submission for doemer
Due date: Mon Oct  8 11:59:59 2007
=====
...

```

Unix System Environment

- Example session (4/4):

```

...
Submit program.c [yes, no]? y
Cannot read file program.c
Submit mybackup.c [yes, no]? n
=====
Summary:
=====
You just submitted file(s):
  program.c
You have not submitted file(s):
  mybackup.c
doemer@vivian% ~eecs10/bin/listfiles.py
=====
EECS 10 Fall 2007: "hw1" listing for doemer
=====
Files submitted for assignment "hw1":
  program.c
doemer@vivian% logout

```

Introduction to Programming

- Categories of programming languages
 - Machine languages (stream of 1's and 0's)
 - Assembly languages (low-level CPU instructions)
 - **High-level languages** (**high-level instructions**)
- Translation of high-level languages
 - Interpreter (translation for each instruction)
 - **Compiler** (**translation once for all code**)
 - Hybrid (combination of the above)
- Types of programming languages
 - Functional (e.g. Lisp)
 - **Structured** (e.g. Pascal, **C**, Ada)
 - Object-oriented (e.g. C++, Java, Python)

History of C

- Evolved from BCPL and B
 - in the 60's and 70's
- Created in 1972 by Dennis Ritchie (Bell Labs)
 - first implementation on DEC PDP-11
 - added concept of *typing* (and other features)
 - development language of UNIX operating system
- “Traditional” C
 - 1978, “*The C Programming Language*”, by Brian W. Kernighan, Dennis M. Ritchie
 - ported to most platforms
- ANSI C
 - standardized in 1989 by ANSI and OSI
 - standard updated in 1999

EECS10: Computational Methods in ECE, Review 1-8

(c) 2008 R. Doemer

15

Introduction to C

- What is C?
 - Programming language
 - high-level
 - structured
 - compiled
 - Standard library
 - rich collection of existing functions
- Why C?
 - de-facto standard in software development
 - code is portable to many different platforms
 - supports structured and functional programming
 - easy transition to object-oriented programming
 - C++ / Java
 - freely available for most platforms

EECS10: Computational Methods in ECE, Review 1-8

(c) 2008 R. Doemer

16

Our first C Program

- Program example: `HelloWorld.c`

```

/* HelloWorld.c: our first C program */
/*                                     */
/* author: Rainer Doemer               */
/*                                     */
/* modifications:                       */
/* 09/28/04 RD initial version         */
/*                                     */

#include <stdio.h>

/* main function */

int main(void)
{
    printf("Hello World!\n");
    return 0;
}

/* EOF */

```

Our first C Program

- Program comments
 - start with `/*` and end with `*/`
 - are ignored by the compiler
 - should be used to
 - document the program code
 - structure the program code
 - enhance the readability
- `#include` preprocessor directive
 - inserts a header file into the code
- standard header file `<stdio.h>`
 - part of the C standard library
 - contains declarations of standard types and functions for data input and output (e.g. function `printf()`)

```

/* HelloWorld.c: our first C program */
/* author: Rainer Doemer               */
/* modifications:                       */
/* 09/28/04 RD initial version         */
/*                                     */
#include <stdio.h>
/* main function */
int main(void)
{
    printf("Hello World!\n");
    return 0;
}
/* EOF */

```

Our first C Program

- **int main(void)**
 - main function of the C program
 - the program execution starts (and ends) here
 - **main** must return an integer (**int**) value to the operating system at the end of its execution
 - return value of 0 indicates successful completion
 - return value greater than 0 usually indicates an error condition
- **function body**
 - block of code (definitions and statements)
 - starts with an opening brace (**{**)
 - ends with a closing brace (**}**)
- **printf()** function
 - formatted output (to **stdout**)
- **return statement**
 - ends a function and returns its argument as result

```
...
/* main function */
int main(void)
{
    printf("Hello World!\n");
    return 0;
}
/* EOF */
```

Our first C Program

- **Program compilation**
 - compiler translates the code into an executable program
 - **gcc HelloWorld.c**
 - compiler reads file **HelloWorld.c** and creates file **a.out**
 - options may be specified to direct the compilation
 - **-o HelloWorld** specifies output file name
 - **-ansi -Wall** specifies ANSI code with all warnings
- **Program execution**
 - use the generated executable as command
 - **HelloWorld**
 - the operating system loads the program (loader), then executes its instructions (program execution), and finally resumes when the program has terminated

Our first C Program

- Example session: HelloWorld.c

```
east% mkdir HelloWorld
east% cd HelloWorld
east% ls
east% vi HelloWorld.c
east% ls
HelloWorld.c
east% ls -l
-rw-r--r--  1 doemer  faculty    263 Sep 28 22:11 HelloWorld.c
east% gcc HelloWorld.c
east% ls -l
-rw-r--r--  1 doemer  faculty    263 Sep 28 22:11 HelloWorld.c
-rwxr-xr-x  1 doemer  faculty   6352 Sep 28 22:12 a.out*
east% a.out
Hello World!
east% gcc -Wall -ansi HelloWorld.c -o HelloWorld
east% ls -l
-rwxr-xr-x  1 doemer  faculty   6356 Sep 28 22:17 HelloWorld*
-rw-r--r--  1 doemer  faculty    263 Sep 28 22:17 HelloWorld.c
-rwxr-xr-x  1 doemer  faculty   6352 Sep 28 22:12 a.out*
east% HelloWorld
Hello World!
```

EECS10: Computational Methods in ECE, Review 1-8

(c) 2008 R. Doemer

21

Our first C Program

- Character string constants: "Strings"
 - start and end with a double quote character (")
 - may not extend over a single line
 - subsequent string constants are combined
 - text formatting using escape sequences
 - `\n` new line
 - `\t` horizontal tab
 - `\r` carriage return
 - `\b` back space
 - `\a` alert / bell
 - `\\` backslash character
 - `\"` double quote character
- Experiments with the `HelloWorld` program...

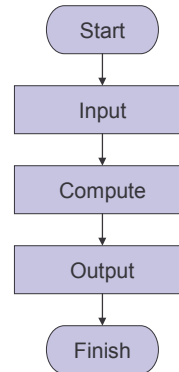
EECS10: Computational Methods in ECE, Review 1-8

(c) 2008 R. Doemer

22

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.



EECS10: Computational Methods in ECE, Review 1-8

(c) 2008 R. Doemer

23

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

EECS10: Computational Methods in ECE, Review 1-8

(c) 2008 R. Doemer

24

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**, **i2**, **sum**

- 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);
```

- part of 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
- May 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);
```

- part of 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
 - standard text is copied verbatim to the output
 - `"%d"` is replaced with a decimal integer value
- takes values from specified arguments
 - `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.
%
```

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, Review 1-8

(c) 2008 R. Doemer

33

Conversion Specifiers for Basic Types

• Type	<code>printf()</code>	<code>scanf()</code>
• <code>long double</code>	<code>%Lf</code>	<code>%Lf</code>
• <code>double</code>	<code>%f</code>	<code>%lf</code>
• <code>float</code>	<code>%f</code>	<code>%f</code>
• <code>unsigned long long</code>	<code>%llu</code>	<code>%llu</code>
• <code>long long</code>	<code>%lld</code>	<code>%lld</code>
• <code>unsigned long</code>	<code>%lu</code>	<code>%lu</code>
• <code>long</code>	<code>%ld</code>	<code>%ld</code>
• <code>unsigned int</code>	<code>%u</code>	<code>%u</code>
• <code>int</code>	<code>%d</code>	<code>%d</code>
• <code>short</code>	<code>%hd</code>	<code>%hd</code>
• <code>char</code>	<code>%c</code>	<code>%c</code>

EECS10: Computational Methods in ECE, Review 1-8

(c) 2008 R. Doemer

34

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 %, <<, >>

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

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

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 Arithmetic.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
%
```

Shift Operators

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

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` type into a `float` type
 - `Integer = (int) Double`
 - converts the `double` type into an `int` type
 - any fractional part is truncated!
 - `Char = (char) LongLongInt`
 - converts the `long long int` type into a `char` type
 - any out-of-range values are silently cut off!

EECS10: Computational Methods in ECE, Review 1-8

(c) 2008 R. Doemer

41

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
 - 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`

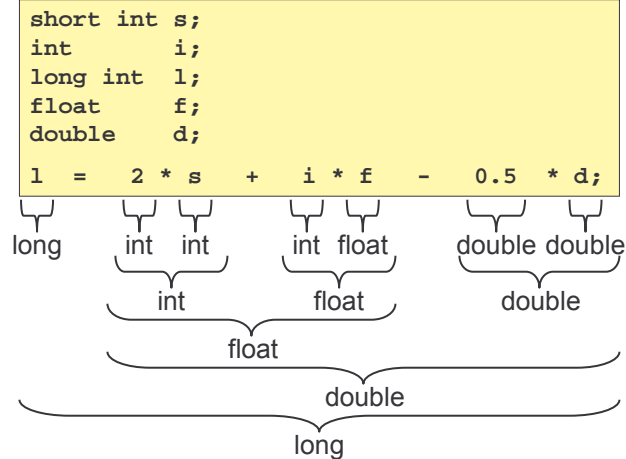
EECS10: Computational Methods in ECE, Review 1-8

(c) 2008 R. Doemer

42

Types in Expressions

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



EECS10: Computational Methods in ECE, Review 1-8

(c) 2008 R. Doemer

43

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:

$$r = \text{rem}(2^n / 7)$$

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

EECS10: Computational Methods in ECE, Review 1-8

(c) 2008 R. Doemer

44

Example Program

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

```

/* Arithmetic.c: arithmetic expressions      */
/*                                           */
/* 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.
%

```


Comparison of Values

- Relational Operators
 - direct comparison of two values
 - Boolean result: truth value, true or false
- Logical Operators
 - Operations on Boolean values
- Conditional Operator
 - Conditional evaluation of expressions

Relational Operators

- Comparison operations
 - < less than
 - > greater than
 - <= less than or equal to
 - >= greater than or equal to
 - == equal to (remember, = means assignment!)
 - != not equal to
- Comparison is defined for all basic types
 - integer (e.g. $5 < 6$)
 - floating point (e.g. $7.0 < 7e1$)
- Result type is Boolean, but represented as integer
 - false 0
 - true 1 (or any other value *not* equal to zero)

Logical Operators

- Operation on Boolean/truth values

- ! “not” logical negation
- && “and” logical and
- || “or” logical or

- Truth table:

x	y	!x	x && y	x y
0	0	1	0	0
0	1	1	0	1
1	0	0	0	1
1	1	0	1	1

- Argument and result types are Boolean, but represented as integer

- false 0
- true 1 (or any other value *not* equal to zero)

Conditional Operator

- Conditional evaluation of values in expressions

- Question-mark operator:

test ? true-value : false-value

- evaluates the *test*
- if *test* is true, then the result is *true-value*
- otherwise, the result is *false-value*

- Examples:

- $(4 < 5) ? (42) : (4+8)$ evaluates to 42
- $(2==1+2) ? (x) : (y)$ evaluates to *y*
- $(x < 0) ? (-x) : (x)$ evaluates to ***abs(x)***

Operator Evaluation Order

- Associativity: left to right or right to left
- Precedence: group-wise, top to bottom
 - parentheses (,) n/a
 - unary plus, minus, negation +, -, ! right to left
 - type casting (*typename*) right to left
 - multiplication, division, modulo *, /, % left to right
 - addition, subtraction +, - left to right
 - shift left, shift right <<, >> left to right
 - relational operators <, <=, >=, > left to right
 - equality ==, != left to right
 - logical and && left to right
 - logical or || left to right
 - conditional operator ?: left to right
 - assignment operator = right to left

Conditional Statements

- **if** statement
 - Control flow statement for decision making
 - Changes control flow depending on a specified condition
 - Example:
 - ```
if (x < 0)
 { printf("%d is negative", x); }
```
    - ```
if (x >= 0)
  { printf("%d is positive", x); }
```
 - Syntax: **if** construct consists of
 - Keyword `if`
 - Condition expression evaluated to true or false
 - Body statement block
 - Semantics:
 - Body is executed *only if* the condition evaluates to true

Example Program

- Comparison of values: `Comparison.c` (part 1/3)

```

/* Comparison.c: arithmetic comparisons      */
/*                                           */
/* author: Rainer Doemer                    */
/*                                           */
/* modifications:                           */
/* 10/07/04 RD initial version              */

#include <stdio.h>

/* main function */

int main(void)
{
    /* variable definitions */
    int a, b;

    ...

```

Example Program

- Comparison of values: `Comparison.c` (part 2/3)

```

...
/* input section */
printf("Please enter a value for integer a: ");
scanf("%d", &a);
printf("Please enter a value for integer b: ");
scanf("%d", &b);

/* computation and output section */
if (a == b)
{ printf("%d is equal to %d.\n", a, b);
} /* fi */
if (a != b)
{ printf("%d is not equal to %d.\n", a, b);
} /* fi */
if (a < b)
{ printf("%d is less than %d.\n", a, b);
} /* fi */
...

```

Example Program

- Comparison of values: `Comparison.c` (part 3/3)

```

...
    if (a > b)
        { printf("%d is greater than %d.\n", a, b);
          } /* fi */
    if (a <= b)
        { printf("%d is less than or equal to %d.\n", a, b);
          } /* fi */
    if (a >= b)
        { printf("%d is greater than or equal to %d.\n", a, b);
          } /* fi */

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

/* EOF */

```

Example Program

- Example session: `Comparison.c`

```

% vi Comparison.c
% gcc -Wall -ansi Comparison.c -o Comparison
% Comparison
Please enter a value for integer a: 42
Please enter a value for integer b: 56
42 is not equal to 56.
42 is less than 56.
42 is less than or equal to 56.
% Comparison
Please enter a value for integer a: 6
Please enter a value for integer b: 6
6 is equal to 6.
6 is less than or equal to 6.
6 is greater than or equal to 6.
% Comparison
Please enter a value for integer a: 77
Please enter a value for integer b: 6
77 is not equal to 6.
...

```

Keywords in C

- List of keywords in ANSI-C

- 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

- These keywords are reserved!
- These cannot be used as identifiers.
- More keywords are reserved for C++

Augmented Assignment Operators

- Assignment operator: =
 - evaluates right-hand side
 - assigns result to left-hand side
- Augmented assignment operators: +=, *=, ...
 - evaluates right-hand side as temporary result
 - applies operation to left-hand side and temporary result
 - assigns result of operation to left-hand side
- Example: Counter
 - `int c = 0; /* counter starting from 0 */`
 - `c = c + 1; /* counting by regular assignment */`
 - `c += 1; /* counting by augmented assignment */`
- Augmented assignment operators:
 - +=, -=, *=, /=, %=, <<=, >>=, |=, &&=

Increment and Decrement Operators

- Counting in steps of one
 - increment (add 1)
 - decrement (subtract 1)
- C provides special operators
 - increment operator: ++
 - `count++` post-increment (`count += 1`)
 - `++count` pre-increment (`count += 1`)
 - decrement operator: --
 - `count--` post-decrement (`count -= 1`)
 - `--count` pre-increment (`count -= 1`)
 - *pre-* increment/decrement
 - value returned is the incremented/decremented (new) value
 - *post-* increment/decrement
 - value returned is the original (old) value

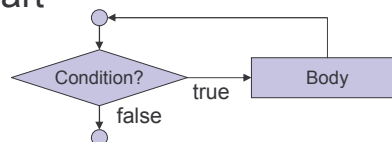
EECS10: Computational Methods in ECE, Review 1-8

(c) 2008 R. Doemer

61

Repetition Statements

- Repetition (aka. iteration, loop)
 - repeated execution of a block of statements
 - counter-controlled
 - counter determines number of repetitions (often predefined at compile time)
 - sentinel-controlled
 - sentinel condition determines number of repetitions (usually determined at run time)
- Control flow chart



EECS10: Computational Methods in ECE, Review 1-8

(c) 2008 R. Doemer

62

Repetition Statements

- **while** loop
 - Control flow statement for repetition (iteration)
 - Repeats execution depending on a specified condition
 - Example:


```
int product = 2;
while (product < 1000)
{ product *= 2; }
printf("Product is %d", product);
```
 - Syntax: **while** construct consists of
 - keyword `while`
 - condition expression evaluated to true or false
 - body statement block
 - Semantics: the body is repeatedly executed as long as the condition evaluates to true
 - the condition is evaluated at the *beginning* of each loop

Example Program

- Average of values: **Average.c** (part 1/3)

```
/* Average.c: compute the average of a set of numbers */
/*
/* author: Rainer Doemer
/*
/* modifications:
/* 10/10/04 RD initial version
*/

#include <stdio.h>

/* main function */

int main(void)
{
    /* variable definitions */
    int counter;
    double value;
    double total;
    double average;
    ...
}
```


Example Program

- Average of values: `Average.c` (part 2/3)

```
...  
  
/* input and computation section */  
counter = 1;  
total = 0.0;  
while (counter <= 10)  
{ printf("Please enter value %d: ", counter);  
  scanf("%lf", &value);  
  total += value;  
  counter++;  
} /* elihw */  
  
/* computation section */  
average = total / 10.0;  
  
...
```

Example Program

- Average of values: `Average.c` (part 3/3)

```
...  
  
/* output section */  
printf("The average is %f.\n", average);  
  
/* exit */  
return 0;  
} /* end of main */  
  
/* EOF */
```

Example Program

- Example session: `Average.c`

```
% vi Average.c
% gcc Average.c -o Average -Wall -ansi
% Average
Please enter value 1: 23
Please enter value 2: 25
Please enter value 3: 17
Please enter value 4: 18.6
Please enter value 5: 50.8
Please enter value 6: 33.3
Please enter value 7: 12
Please enter value 8: 42
Please enter value 9: 42.2
Please enter value 10: 34
The average is 29.790000.
%
```

Repetition Statements

- Explicit control flow in loops
 - `break` statement
 - exits the innermost loop
 - `continue` statement
 - jump back to the beginning of the innermost loop

- Example:

```
int i = 0;
int s = 0;
while (1)          /* "endless" loop */
{
    i++;
    if (i > 10)
        { break; } /* exit the loop */
    if (i % 2 == 1)
        { continue; } /* next iteration */
    s += i;
} /* elihw */
printf("%d", s);
```

Example Program

- Average of values: `Average2.c` (part 1/3)

```

/* Average2.c: compute the average of a set of numbers */
/*
/* author: Rainer Doemer
/*
/*
/* modifications:
/* 10/10/04 RD sentinel controlled loop
/* 10/10/04 RD initial version
*/

#include <stdio.h>

/* main function */

int main(void)
{
    /* variable definitions */
    int counter;
    double value;
    double total;
    double average;
    ...

```

Example Program

- Average of values: `Average2.c` (part 2/3)

```

...

/* input and computation section */
counter = 0;
total = 0.0;
while (1)
{ printf("Please enter a value (or -1 to quit): ");
  scanf("%lf", &value);
  if (value == -1.0)
  { break;
  } /* fi */
  total += value;
  counter++;
} /* elihw */

...

```

Example Program

- Average of values: `Average2.c` (part 3/3)

```

...

/* computation and output section */
printf("%d values entered.\n", counter);
if (counter >= 1)
{ average = total / (double)counter;
  printf("The average is %f.\n", average);
} /* fi */

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

/* EOF */

```

Example Program

- Example session: `Average2.c`

```

% vi Average2.c
% gcc Average2.c -o Average2 -Wall -ansi
% Average2
Please enter a value (or -1 to quit): 2
Please enter a value (or -1 to quit): 3
Please enter a value (or -1 to quit): 4
Please enter a value (or -1 to quit): 5
Please enter a value (or -1 to quit): -1
4 values entered.
The average is 3.500000.
% Average2
Please enter a value (or -1 to quit): -1
0 values entered.
%

```