EECS 10: Computational Methods in Electrical and Computer Engineering

Lecture 5

Rainer Dömer

doemer@uci.edu

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

Lecture 5: Overview

- Review Quiz
- Keywords in C
- Comparison of Values
  - Relational, logical, and conditional operators
- Conditional Statements
  - **`if`** statement
  - Example **`Comparison.c`**
- Counters
- Repetition Statements
  - **`while`** loop
  - Example **`Average.c`**

Quiz: Question 11

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

```
x = 3 << 2 >> 1;
```

a) **Syntax Error!**
b) **3**
c) **6**
d) **12**
e) **321**

Quiz: Question 11

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

```
x = 3 << 2 >> 1;
```

a) Syntax Error!
b) 3
c) **6**
d) 12
e) 321

Quiz: Question 12

- Which of the following constants is of type **`double`**?
  (Check all that apply!)
  a) **`42`**
  b) **`.42`**
  c) **`4e2`**
  d) **`4E2`**
  e) **`42f`**

Quiz: Question 12

- Which of the following constants is of type **`double`**?
  (Check all that apply!)
  a) **`42`**
  b) **`.42`**
  c) **`4e2`**
  d) **`4E2`**
  e) **`42f`**

Quiz: Question 13

- What is the result type of the following expression?

```
-1 + 2.3f * (4.5 / 67f) - (short)89
```

a) `short int`
b) `int`
c) `long int`
d) `float`
e) `double`

Quiz: Question 13

- What is the result type of the following expression?

```
-1 + 2.3f * (4.5 / 67f) - (short)89
```

a) `short int`
b) `int`
c) `long int`
d) `float`
e) `double`

Quiz: Question 14

- What is the value of **x** after the following code segment?

```
int    i = 10;
double d = 0.5;
double x;

x = i/3 + d;
```

a) 0.333333
b) 3.0
c) 3.333333
d) 3.5
e) 3.833333

Quiz: Question 14

- What is the value of **x** after the following code segment?

```
int    i = 10;
double d = 0.5;
double x;

x = i/3 + d;
```

a) 0.333333
b) 3.0
c) 3.333333
d) 3.5
e) 3.833333

Quiz: Question 15

- Given the following code fragment,

```
double x;
double y;

x = (int)(y + 0.5);
```

which of the following statements is true? (Check all that apply!)

a) for `y=5.0`, `x` is set to `5.0`
b) for `y=5.1`, `x` is set to `5.0`
c) for `y=5.49`, `x` is set to `5.0`
d) for `y=5.5`, `x` is set to `6.0`
e) for `y=5.95`, `x` is set to `6.0`

Quiz: Question 15

- Given the following code fragment,

```
double x;
double y;

x = (int)(y + 0.5);
```

which of the following statements is true? (Check all that apply!)

a) for `y=5.0`, `x` is set to `5.0`
b) for `y=5.1`, `x` is set to `5.0`
c) for `y=5.49`, `x` is set to `5.0`
d) for `y=5.5`, `x` is set to `6.0`
e) for `y=5.95`, `x` is set to `6.0`

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

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)`**

EECS10: Computational Methods in ECE, Lecture 5 (c) 2018 R. Doemer 17

Operator Evaluation Order

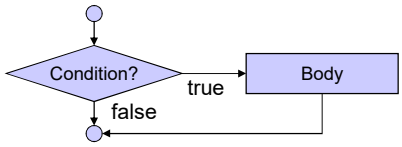
- Associativity: left to right or right to left
- Precedence: group-wise, top to bottom

– 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

EECS10: Computational Methods in ECE, Lecture 5 (c) 2018 R. Doemer 18

Conditional Statements

- **`if`** statement
  - Control flow statement for decision making
    - Changes control flow depending on a specified condition
  - Control flow chart:



  - Semantics:
    - Body is executed *only if* the condition evaluates to true

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

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.
...
```

Augmented Assignment Operators

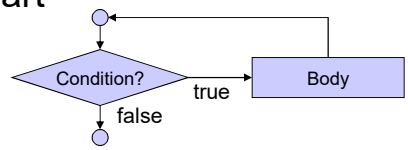
- Assignment operator: `=`
  - evaluates right-hand side
  - assigns result to left-hand side
- Augmented assignment operators: `+=`, `*=`, ...
  - evaluates right-hand side as temporary result
  - applies operation to left-hand side and temporary result
  - assigns result of operation to left-hand side
- 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-decrement (`count -= 1`)
  - *pre-* increment/decrement
    - value returned is the incremented/decremented (new) value
  - *post-* increment/decrement
    - value returned is the original (old) value

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



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.
%
```