EECS 22: Advanced C Programming

Lecture 3

Rainer Dömer

doemer@uci.edu

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

Lecture 3: Overview

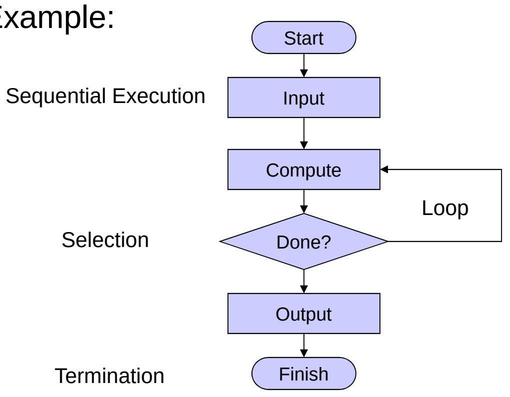
- Review of the C Programming Language
  - Control Flow Charts
  - Structured Programming
    - Sequential statements
    - Conditional statements
    - Repetition statements
    - Arbitrary jump statements
  - Structured Program Composition
  - Example **`Average.c`**

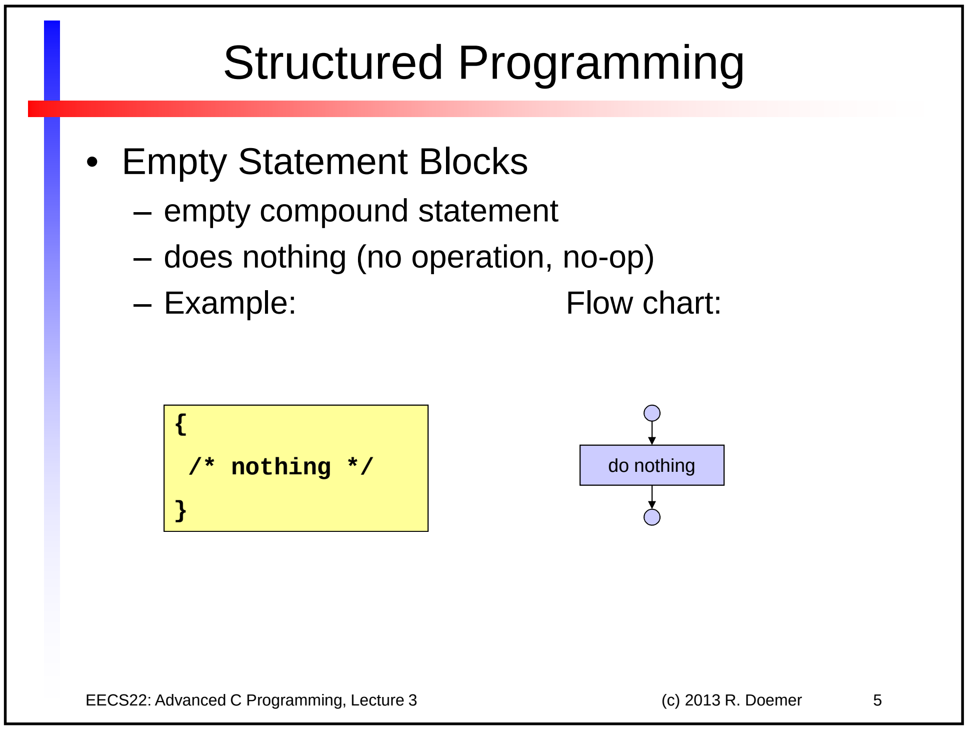
Structured Programming

- Control Flow Statements
  - Sequential execution
    - Compound statements
  - Conditional execution
    - **`if`** statement
    - **`if-else`** statement
    - **`switch`** statement
  - Iterative execution
    - **`while`** loop
    - **`do-while`** loop
    - **`for`** loop
  - Unstructured execution
    - **`goto`** statement

Structured Programming

- Control Flow Chart
  - Graphical representation of program control flow
  - Example:

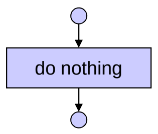


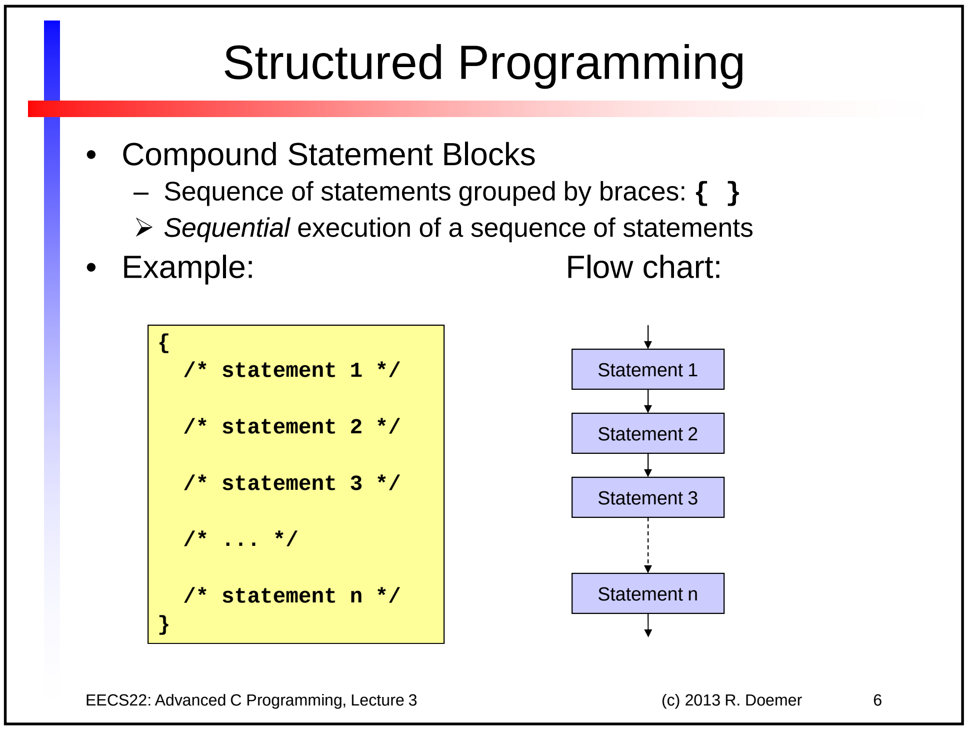


Structured Programming

- Empty Statement Blocks
  - empty compound statement
  - does nothing (no operation, no-op)
  - Example: Flow chart:

```
{
 /* nothing */
}
```

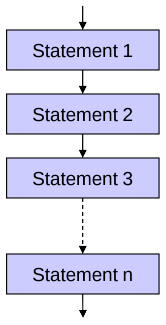




Structured Programming

- Compound Statement Blocks
  - Sequence of statements grouped by braces: **{ }**
  - ➢ *Sequential* execution of a sequence of statements
- Example: Flow chart:

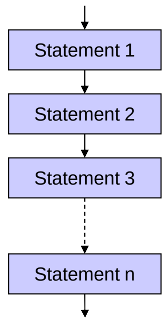
```
{
  /* statement 1 */

  /* statement 2 */

  /* statement 3 */

  /* ... */

  /* statement n */
}
```



Structured Programming

- Compound Statement Blocks
  - Sequence of statements grouped by braces: **{ }**
  - ➢ Compound statements may have *local variables*!
- Example: Flow chart:

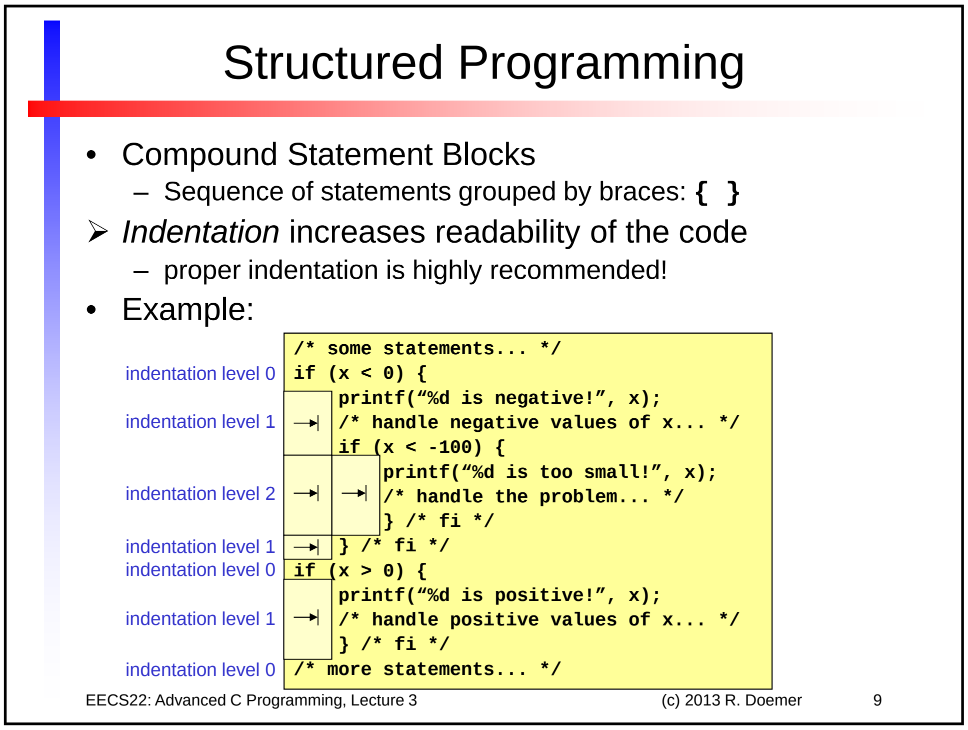
```
{ /* declarations */
  /* definitions  */

  /* statement 1 */
  /* statement 2 */
  /* statement 3 */

  /* ... */

  /* statement n */
}
```



Structured Programming

- Compound Statement Blocks
  - Sequence of statements grouped by braces: **{ }**
- ➢ *Indentation* increases readability of the code
  - proper indentation is highly recommended!
- Example:

```
/* some statements... */
if (x < 0) {
    printf(“%d is negative!”, x);
    /* handle negative values of x... */
    if (x < -100) {
        printf(“%d is too small!”, x);
        /* handle the problem... */
        } /* fi */
    } /* fi */
if (x > 0) {
    printf(“%d is positive!”, x);
    /* handle positive values of x... */
    } /* fi */
/* more statements... */
```

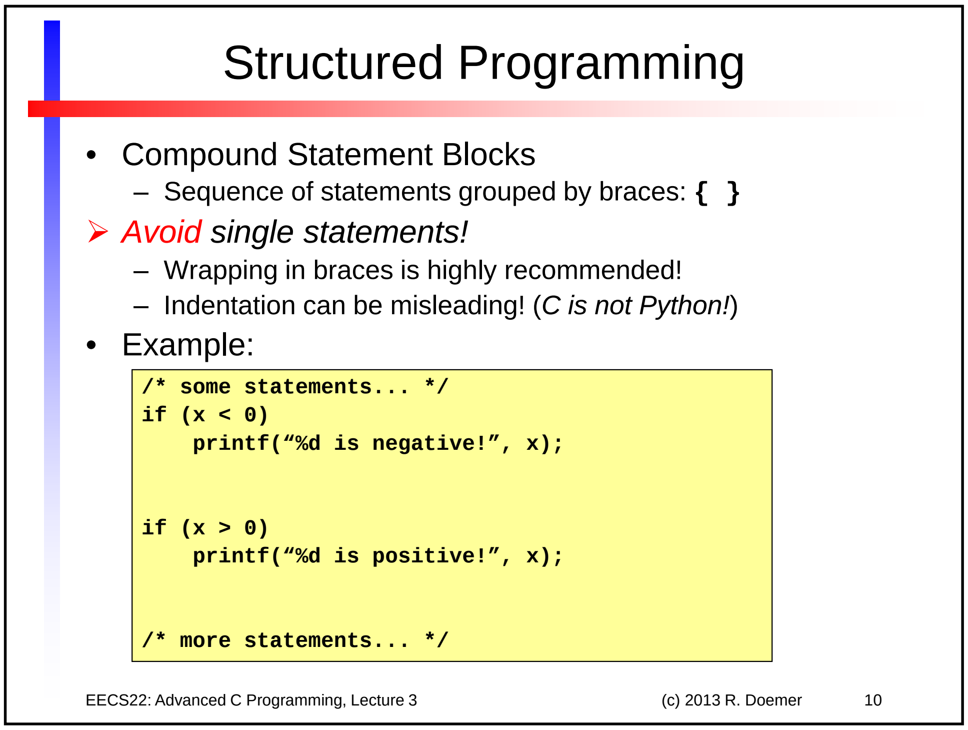


Structured Programming

- Compound Statement Blocks
  - Sequence of statements grouped by braces: **{ }**
- ➢ *Indentation* increases readability of the code
  - proper indentation is highly recommended!
- Example:

```
                       /* some statements... */
indentation level 0    if (x < 0) {
                           printf("%d is negative!", x);
indentation level 1    →|  /* handle negative values of x... */
                           if (x < -100) {
                               printf("%d is too small!", x);
indentation level 2    →|  →|  /* handle the problem... */
                               } /* fi */
indentation level 1    →|  } /* fi */
indentation level 0    if (x > 0) {
                           printf("%d is positive!", x);
indentation level 1    →|  /* handle positive values of x... */
                           } /* fi */
indentation level 0    /* more statements... */
```

EECS22: Advanced C Programming, Lecture 3 (c) 2013 R. Doemer 9



Structured Programming

- Compound Statement Blocks
  - Sequence of statements grouped by braces: **{ }**
- ➢ *Avoid single statements!*
  - Wrapping in braces is highly recommended!
  - Indentation can be misleading! (*C is not Python!*)
- Example:

```
/* some statements... */
if (x < 0)
    printf("%d is negative!", x);


if (x > 0)
    printf("%d is positive!", x);


/* more statements... */
```

EECS22: Advanced C Programming, Lecture 3 (c) 2013 R. Doemer 10

Structured Programming

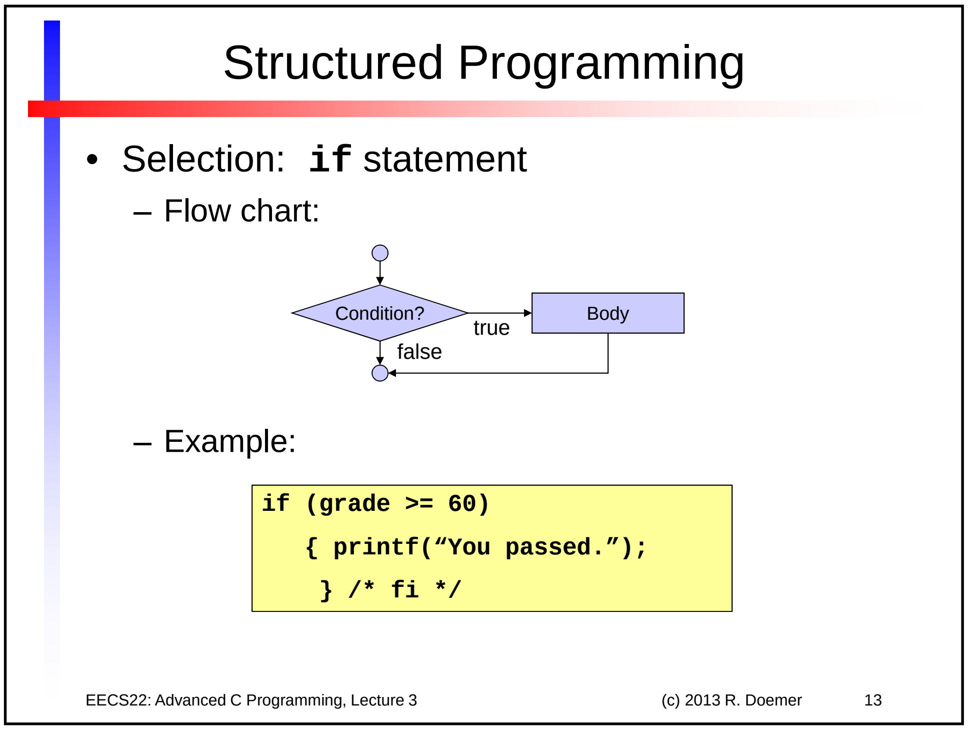
- Compound Statement Blocks
  - Sequence of statements grouped by braces: **{ }**
- ➢ *Avoid single statements!*
  - Wrapping in braces is highly recommended!
  - Indentation can be misleading! (*C is not Python!*)
- Example:

```
/* some statements... */
if (x < 0)
    printf("%d is negative!", x);
    y = sqrt(-x);  /* ERROR! */

if (x > 0)
    printf("%d is positive!", x);
    y = sqrt(x);   /* ERROR! */

/* more statements... */
```

Structured Programming

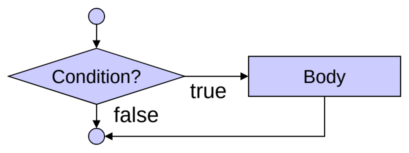
- Compound Statement Blocks
  - Sequence of statements grouped by braces: **{ }**
- ➢ *Avoid single statements!*
  - Wrapping in braces is highly recommended!
  - Indentation can be misleading! (*C is not Python!*)
- Example:

```
/* some statements... */
if (x < 0) {
    printf("%d is negative!", x);
    y = sqrt(-x);
    } /* fi */
if (x > 0) {
    printf("%d is positive!", x);
    y = sqrt(x);
    } /* fi */
/* more statements... */
```



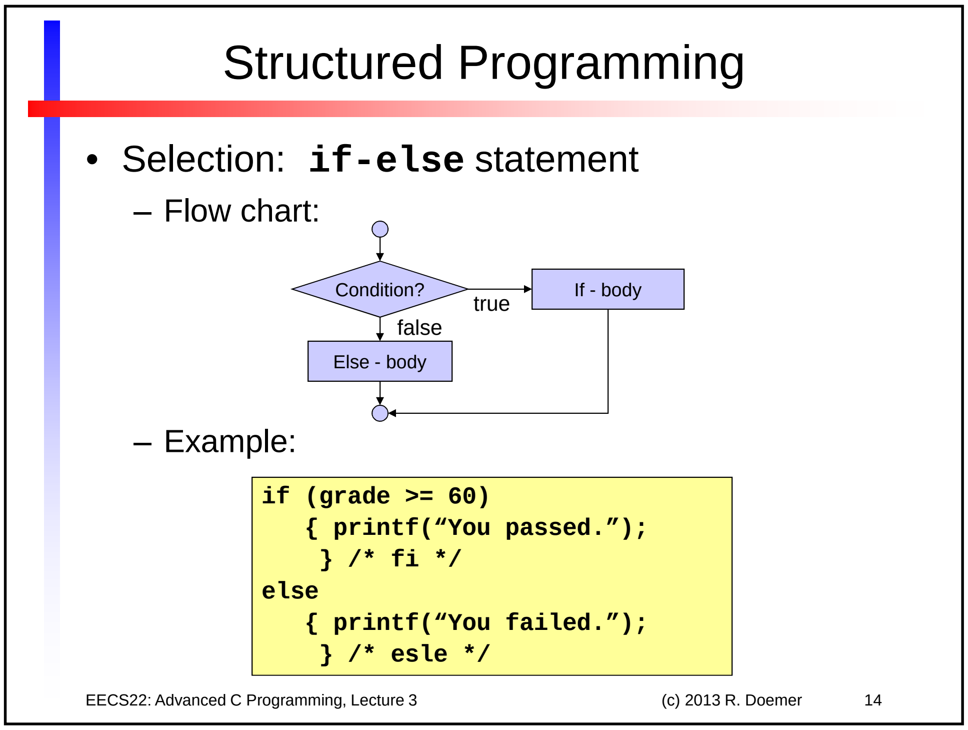
Structured Programming

- Selection: **`if`** statement
  - Flow chart:



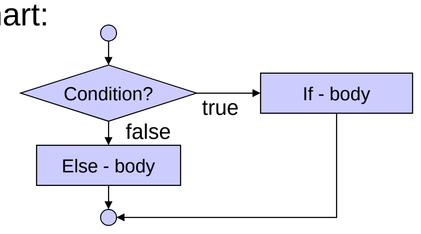
  - Example:

```
if (grade >= 60)
   { printf(“You passed.”);
    } /* fi */
```



Structured Programming

- Selection: **`if-else`** statement
  - Flow chart:

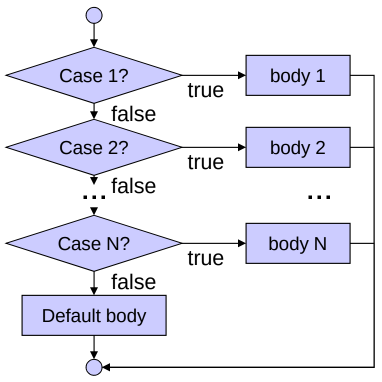


  - Example:

```
if (grade >= 60)
   { printf(“You passed.”);
    } /* fi */
else
   { printf(“You failed.”);
    } /* esle */
```

Structured Programming

- Selection: **`switch`** statement
  - Flow chart:



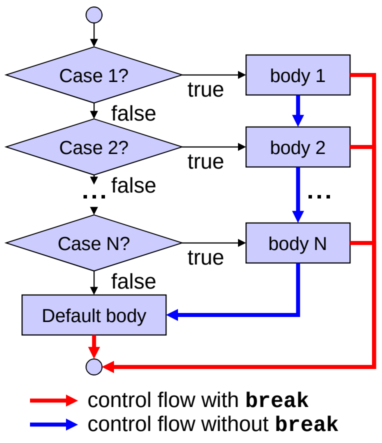
Example:

```
switch(LetterGrade)
{ case 'A':
   { printf("Excellent!");
     break; }
  case 'B':
  case 'C':
  case 'D':
   { printf("Passed.");
     break; }
  case 'F':
   { printf("Failed!");
     break; }
  default:
   { printf("Invalid grade!");
     break; }
} /* hctiws */
```

EECS22: Advanced C Programming, Lecture 3   (c) 2013 R. Doemer   15

Structured Programming

- Selection: **`break`** in **`switch`** statement
  - Flow chart:



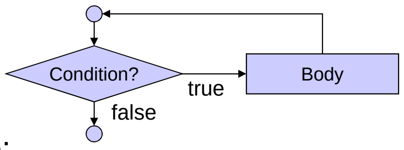
Example:

```
switch(LetterGrade)
{ case 'A':
   { printf("Excellent!");
     break; }
  case 'B':
  case 'C':
  case 'D':
   { printf("Passed.");
     break; }
  case 'F':
   { printf("Failed!");
     break; }
  default:
   { printf("Invalid grade!");
     break; }
} /* hctiws */
```

EECS22: Advanced C Programming, Lecture 3   (c) 2013 R. Doemer   16

Structured Programming

- Repetition: **`while`** loop
  - Flow chart:



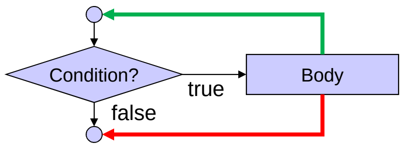
  - Example:

```
int product = 2;
while (product < 1000)
   { product *= 2;
    } /* elihw */
```

  - Note:
    - The condition is evaluated at the *beginning* of each loop!

Structured Programming

- Repetition: **`break`**/**`continue`** in **`while`** loop
  - Flow chart:



  - Control flow:

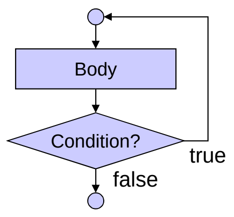
🟥→ control flow with **`break`**

🟩→ control flow with **`continue`**

  - Note:
    - The condition is evaluated at the *beginning* of each loop!

Structured Programming

- Repetition: **`do-while`** loop
  - Flow chart:



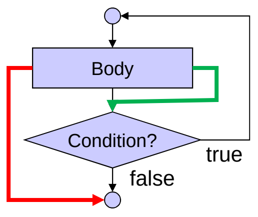
  - Example:

```
int product = 2;
do { product *= 2;
    } while (product < 1000);
```

  - Note:
    - The condition is evaluated at the *end* of each loop!

Structured Programming

- Repetition: **`break`**/**`continue`** in **`do-while`** loop
  - Flow chart:



  - Control flow:

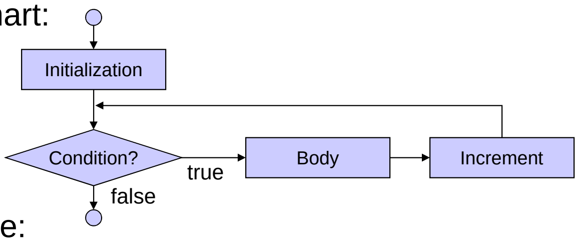
→ control flow with **`break`**

→ control flow with **`continue`**

  - Note:
    - The condition is evaluated at the *end* of each loop!

Structured Programming

- Repetition: **`for`** loop
  - Flow chart:



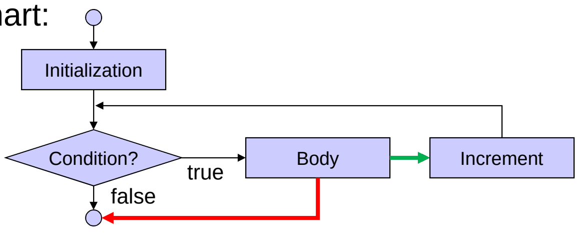
  - Example:

```
for(i = 0; i < 10; i++)
   { printf("i = %d\n", i);
    } /* rof */
```

  - Syntax:
    - **`for(`*`initialization`*`; `*`condition`*`; `*`increment`*`)`**
      `{ `*`body`*` }`

Structured Programming

- Repetition: **`break`**/**`continue`** in **`for`** loop
  - Flow chart:



  - Control flow:

    → control flow with **`break`**

    → control flow with **`continue`**

  - Syntax:
    - **`for(`*`initialization`*`; `*`condition`*`; `*`increment`*`)`**
      `{ `*`body`*` }`

Arbitrary Control Flow

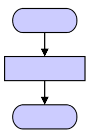
- Arbitrary jumps: **`goto`** statement
  - **`goto`** statement jumps to the specified *labeled* statement (within the same function)
  - Example:

```
begin:  count = 0;
        goto next;
repeat: if (count > 100)
          { goto end; }
next:   count++;
        if (count == 77)
          { goto next; }
        goto repeat;
end:    printf("%d", count);
```

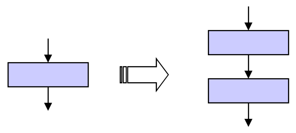
  - Warning:
    - **`goto`** statement allows *un-structured* programming!
    - **`goto`** statement should be avoided whenever possible!

Structured Program Composition

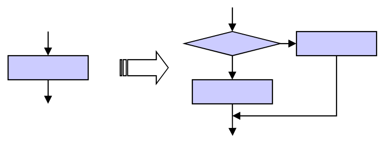
- Initial flow chart
  - Start
  - Program body
  - Finish

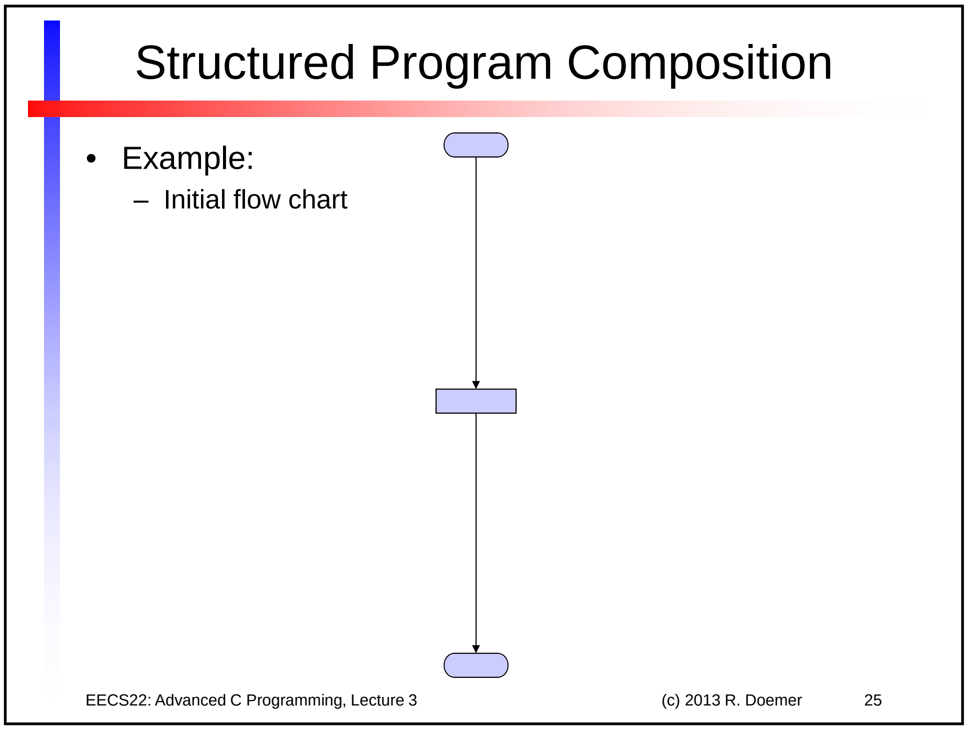


- Statement sequences
  - Statement blocks can be concatenated
  - Sequential execution



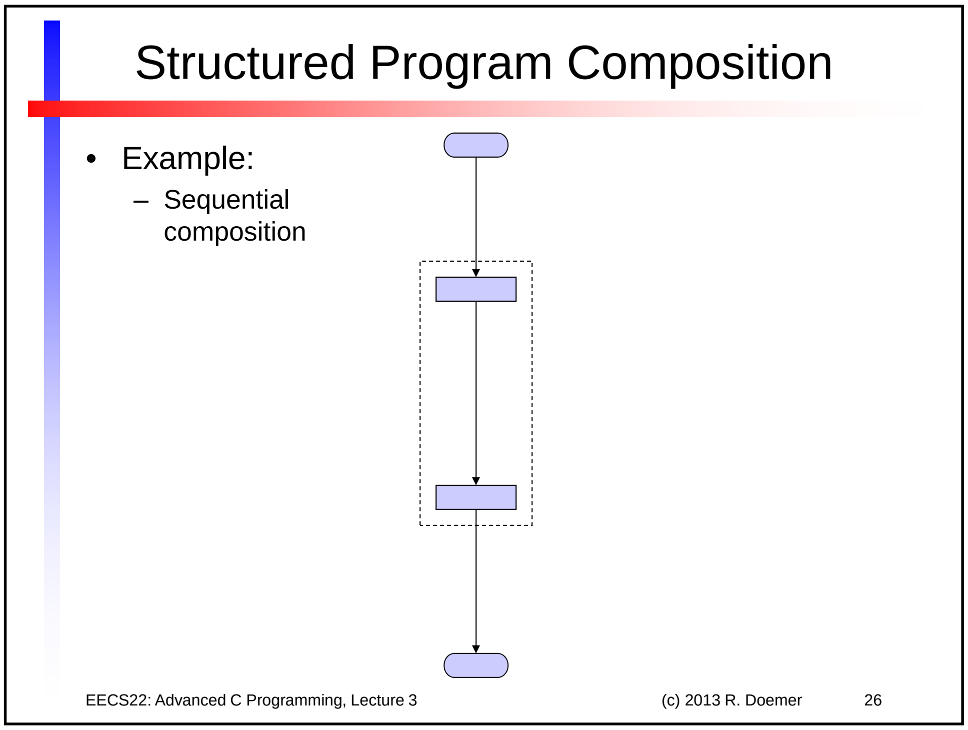
- Nested control structures
  - control structures can be placed wherever statement blocks can be placed in the code





Structured Program Composition

- Example:
  - Initial flow chart

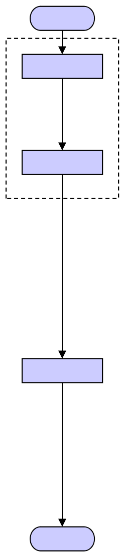


Structured Program Composition

- Example:
  - Sequential composition

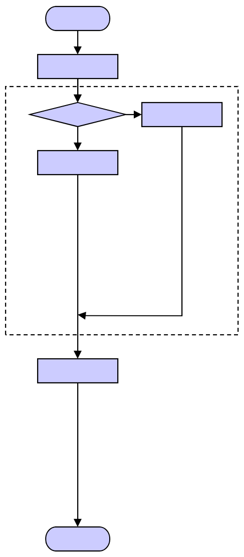
Structured Program Composition

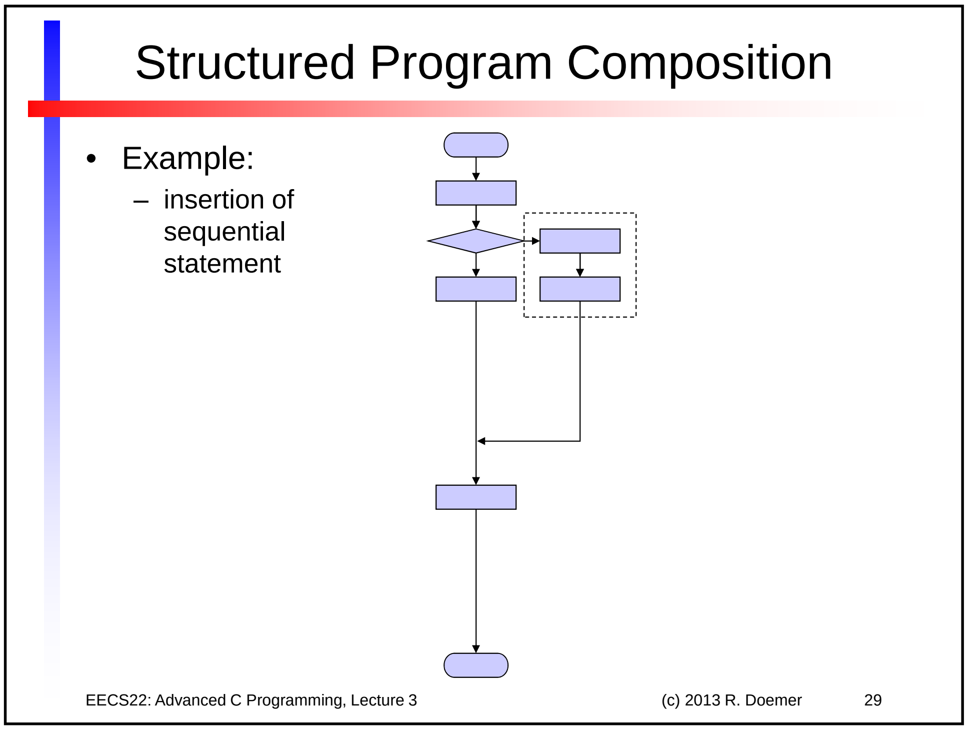
- Example:
  - insertion of another sequential statement



Structured Program Composition

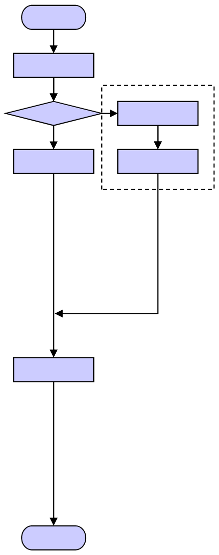
- Example:
  - insertion of **`if – else`** statement

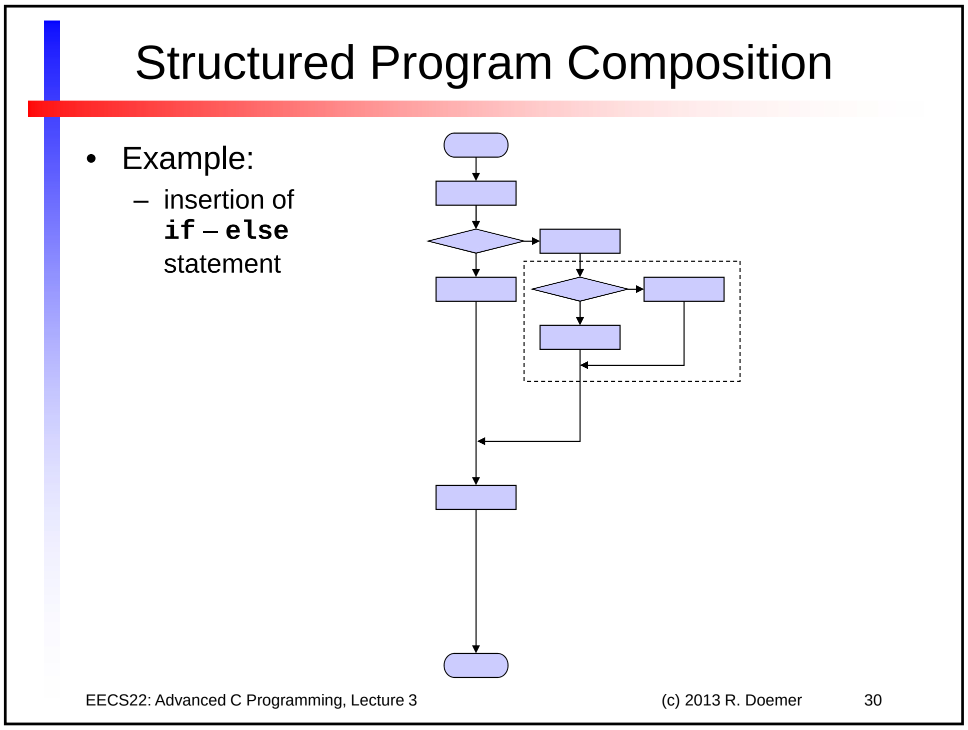




Structured Program Composition

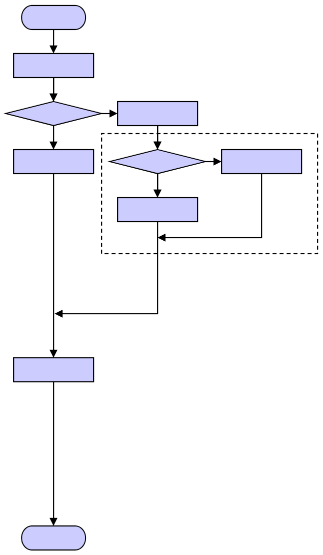
- Example:
  - insertion of sequential statement

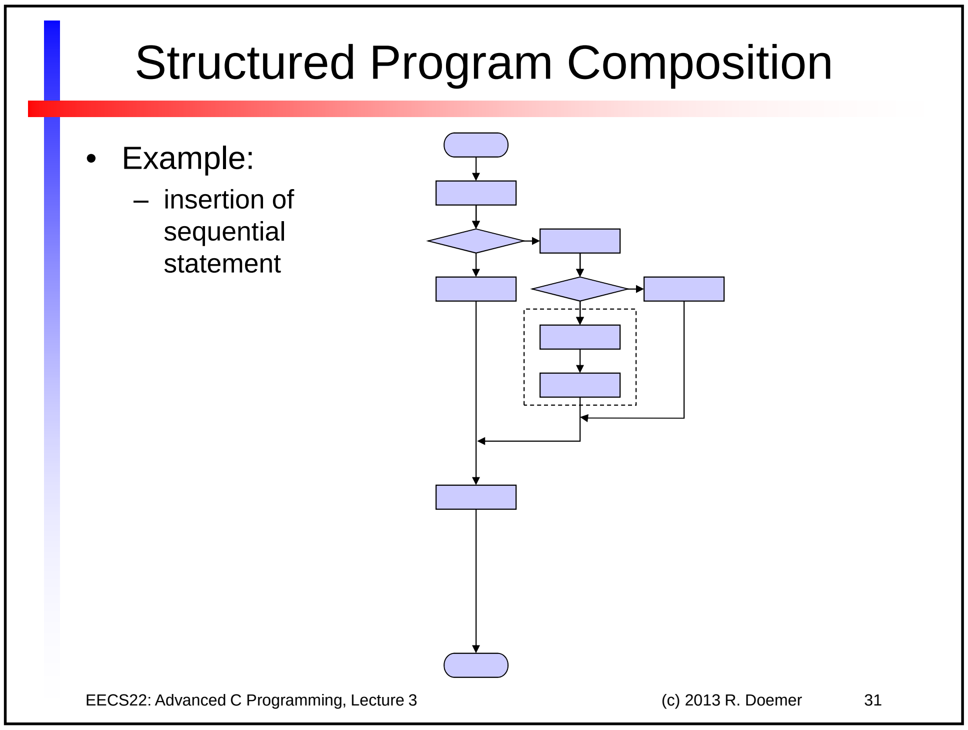




Structured Program Composition

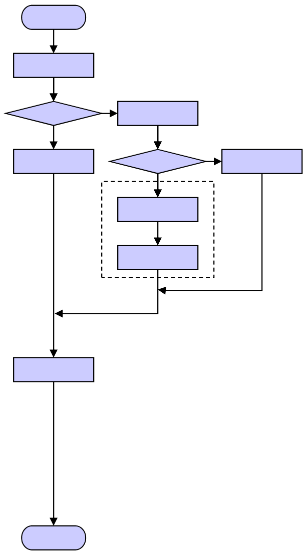
- Example:
  - insertion of **`if – else`** statement

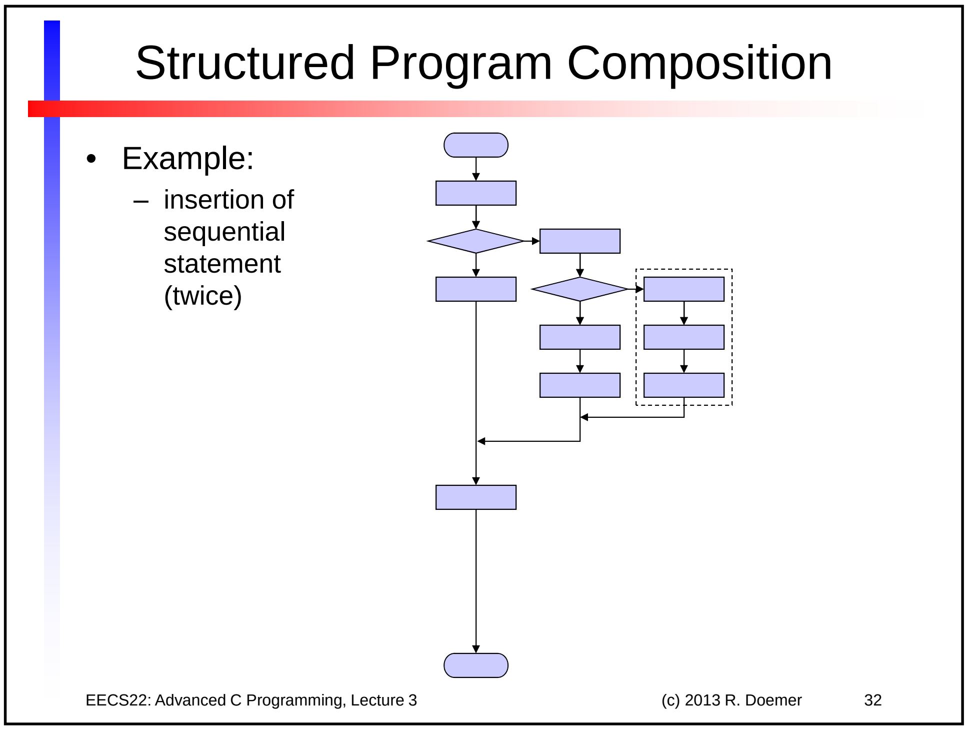




Structured Program Composition

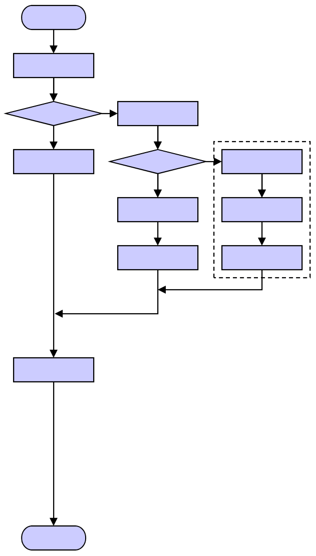
- Example:
  - insertion of sequential statement





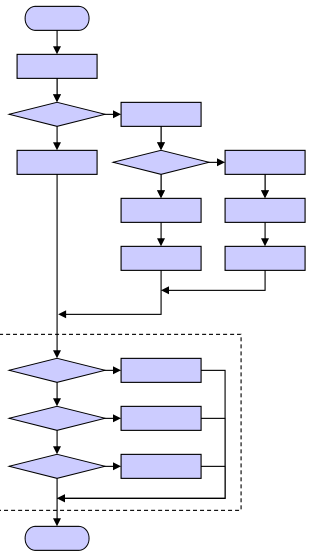
Structured Program Composition

- Example:
  - insertion of sequential statement (twice)



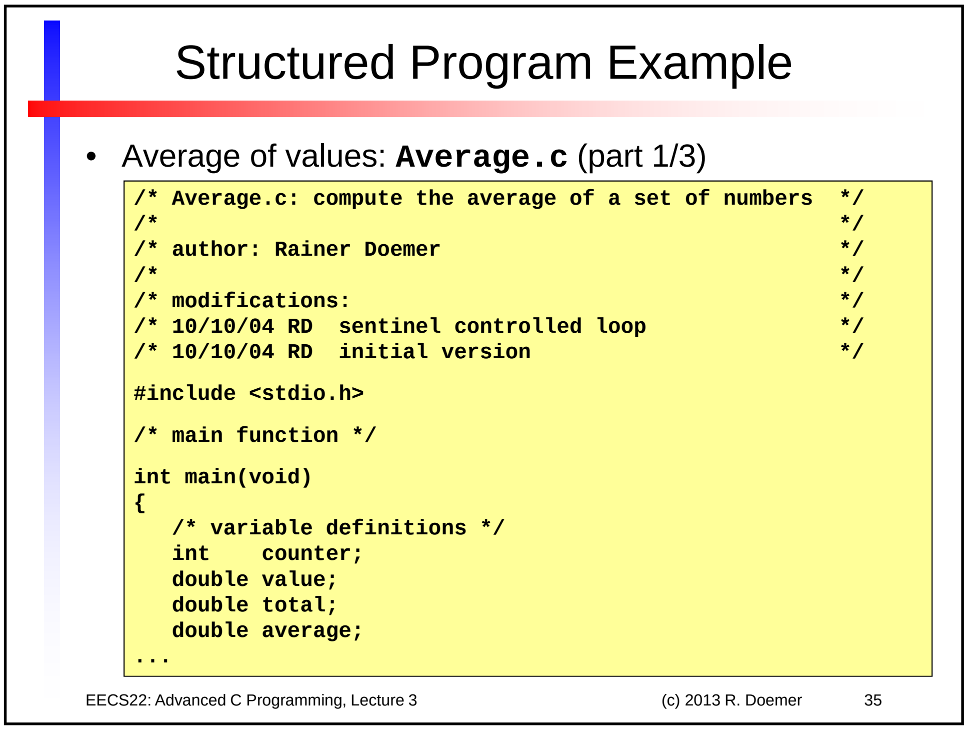
Structured Program Composition

- Example:
  - insertion of **`switch`** statement
  - etc. ...



Structured Program Example

- Example **`Average.c`**
- Task:
  - Compute the average of a set of floating point values
  - The user enters the values consecutively
  - The user enters **`-1`** when done
    - ➢ Sentinel-controlled repetition
  - Print the number of values entered and the calculated average
- Caution:
  - The average of 0 values is undefined!

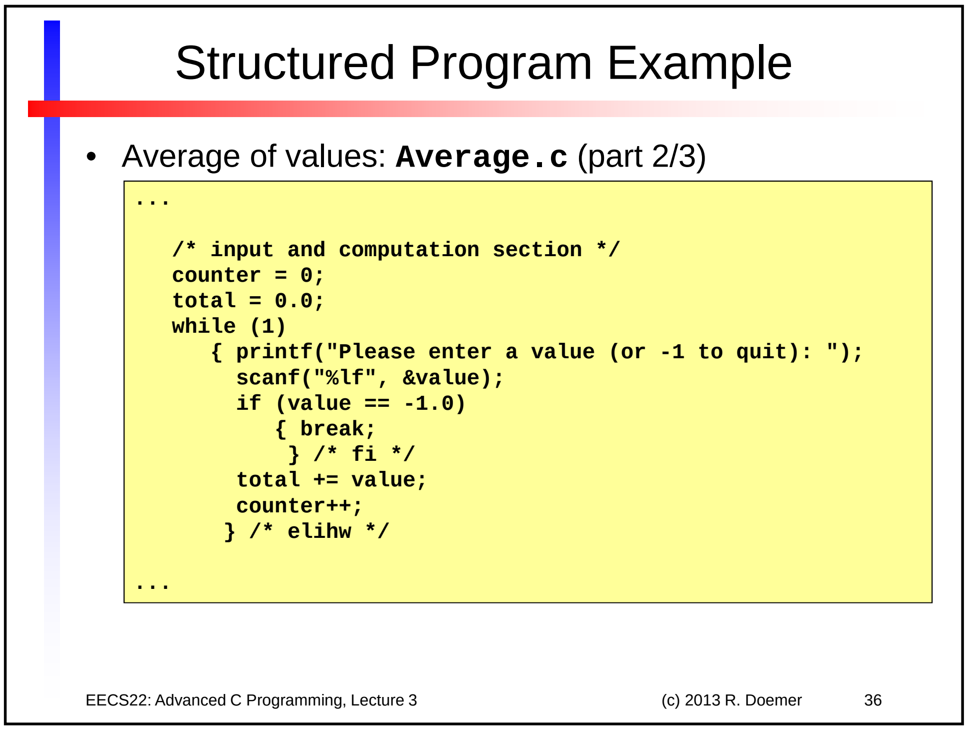


Structured Program Example

- 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  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;
...
```

EECS22: Advanced C Programming, Lecture 3 (c) 2013 R. Doemer 35



Structured Program Example

- Average of values: **`Average.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 */

...
```

EECS22: Advanced C Programming, Lecture 3 (c) 2013 R. Doemer 36

Structured Program Example

- Average of values: **`Average.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 */
```

Structured Program Example

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

```
% vi Average.c
% gcc Average.c -o Average -Wall -ansi
% Average
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.
% Average
Please enter a value (or -1 to quit): -1
0 values entered.
%
```