EECS 10: Computational Methods in Electrical and Computer Engineering

Lecture 7

Rainer Dömer

doemer@uci.edu

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

Lecture 7: Overview

- Formatted Input and Output
- Programming Principles
  - Algorithm and control flow
- Structured Programming
  - Control flow chart
  - Sequential execution
  - Conditional execution
    - **`if`** statement
    - **`if-else`** statement
    - **`switch`** statement
  - Structured Program Composition
  - Examples **`Grade.c`**, **`Grade2.c`**

Formatted Input and Output

- Formatted Input
  - Format specifiers for **`scanf()`**
  - Detailed formatting of integral values
  - Detailed formatting of floating-point values
- Formatted Output
  - Format specifiers for **`printf()`**
  - Detailed formatting of integral values
  - Detailed formatting of floating-point values
- Example **`Formatting.c`**

Formatted Input

- Formatted input using **`scanf()`**
  - standard format specifier for integral values
    - (**`unsigned`**) **`long long`** **`%llu`** **`%lld`**
    - (**`unsigned`**) **`long`** **`%lu`** **`%ld`**
    - (**`unsigned`**) **`int`** **`%u`** **`%d`**
    - (**`unsigned`**) **`short`** **`%hu`** **`%hd`**
    - (**`unsigned`**) **`char`** **`%c`** (reads a character)
  - standard format specifier for floating point values
    - **`long double`** **`%Lf`**
    - **`double`** **`%lf`**
    - **`float`** **`%f`**

Formatted Output

- Formatted output using **`printf()`**
  - standard format specifier for integral values
    - (**`unsigned`**) **`long long`** **`%llu`** **`%lld`**
    - (**`unsigned`**) **`long`** **`%lu`** **`%ld`**
    - (**`unsigned`**) **`int`** **`%u`** **`%d`**
    - (**`unsigned`**) **`short`** **`%hu`** **`%hd`**
    - (**`unsigned`**) **`char`** **`%c`** (prints a character)
  - standard format specifier for floating point values
    - **`long double`** **`%Lf`**
    - **`double`** **`%f`**
    - **`float`** **`%f`**

EECS10: Computational Methods in ECE, Lecture 7 (c) 2019 R. Doemer 5

Formatted Output

- *Detailed* formatting sequence for integral values
  - **`% flags width length conversion`**
  - ***`flags`***
    - (none) standard formatting (right-justified)
    - **`-`** left-justified output
    - **`+`** leading plus-sign for positive values
    - **`0`** leading zeros
  - field ***`width`***
    - (none) minimum number of characters needed
    - integer width of field to be filled with output
  - ***`length`*** modifier
    - (none) **`int`** type
    - **`h`** **`short int`** type
    - **`l`** **`long int`** type
    - **`ll`** **`long long int`** type
  - ***`conversion`*** specifier
    - **`d`** signed decimal value
    - **`u`** unsigned decimal value
    - **`o`** (unsigned) octal value
    - **`x`** (unsigned) hexadecimal value using characters **`0-9`**, **`a-f`**
    - **`X`** (unsigned) hexadecimal value using characters **`0-9`**, **`A-F`**

EECS10: Computational Methods in ECE, Lecture 7 (c) 2019 R. Doemer 6

Formatted Output

- *Detailed* formatting sequence for floating-point values
  - **`% flags width precision length conversion`**
  - **`flags`**
    - (none) standard formatting (right-justified)
    - **`-`** left-justified output
    - **`+`** leading plus-sign for positive values
    - **`0`** leading zeros
  - field **`width`**
    - (none) minimum number of characters needed
    - integer width of field to be filled with output
  - **`precision`**
    - (none) default precision (e.g. 6)
    - **`.`**int number of digits after decimal point (for **`f`**, **`e`**, or **`E`**), maximum number of significant digits (for **`g`**, or **`G`**)
  - **`length`** modifier
    - (none) **`float`** or **`double`** type
    - **`L`** **`long double`** type
  - **`conversion`** specifier
    - **`f`** standard floating-point notation (fixed-point)
    - **`e`** or **`E`** exponential notation (using **`e`** or **`E`**)
    - **`g`** or **`G`** standard or exponential notation (using **`e`** or **`E`**)

Formatted Output

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

```
/* Formatting.c: formatted output demo          */
/* author: Rainer Doemer                        */
/* modifications:                               */
/* 10/19/04 RD  initial version                 */

#include <stdio.h>

/* main function */

int main(void)
{
  /* output section */
  printf("42 formatted as |%%d|:   |%d|\n", 42);
  printf("42 formatted as |%%8d|:  |%8d|\n", 42);
  printf("42 formatted as |%%-8d|: |%-8d|\n", 42);
  printf("42 formatted as |%%+8d|: |%+8d|\n", 42);
  printf("42 formatted as |%%08d|: |%08d|\n", 42);
  printf("42 formatted as |%%x|:   |%x|\n", 42);
  printf("42 formatted as |%%o|:   |%o|\n", 42);
...
```

Formatted Output

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

```
...
  printf("\n");
  printf("123.456 formatted as |%%f|:     |%f|\n", 123.456);
  printf("123.456 formatted as |%%e|:     |%e|\n", 123.456);
  printf("123.456 formatted as |%%g|:     |%g|\n", 123.456);
  printf("123.456 formatted as |%%12.4f|: |%12.4f|\n",
                                                   123.456);
  printf("123.456 formatted as |%%12.4e|: |%12.4e|\n",
                                                   123.456);
  printf("123.456 formatted as |%%12.4g|: |%12.4g|\n",
                                                   123.456);

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

/* EOF */
```


Formatted Output

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

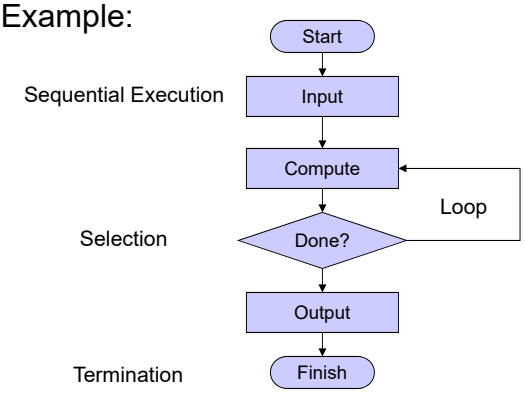
```
% vi Formatting.c
% gcc Formatting.c -o Formatting -Wall -ansi
% ./Formatting
42 formatted as |%d|:   |42|
42 formatted as |%8d|:  |      42|
42 formatted as |%-8d|: |42      |
42 formatted as |%+8d|: |     +42|
42 formatted as |%08d|: |00000042|
42 formatted as |%x|:   |2a|
42 formatted as |%o|:   |52|

123.456 formatted as |%f|:     |123.456000|
123.456 formatted as |%e|:     |1.234560e+02|
123.456 formatted as |%g|:     |123.456|
123.456 formatted as |%12.4f|: |    123.4560|
123.456 formatted as |%12.4e|: |  1.2346e+02|
123.456 formatted as |%12.4g|: |       123.5|
%
```

Programming Principles

- Thorough *understanding* of the problem
- *Problem definition*
  - Input data
  - Output data
- *Algorithm*: Procedure to solve the problem
  - Detailed set of *actions* to perform
  - Specification of *order* in which to perform the actions
  - Termination after a *finite* number of steps
- *Pseudo code*: Planning a program
  - Informal (English) description of steps in an algorithm
  - Example: Cake baking recipe
- *Control flow*
  - Detailed execution order of steps in the program
- *Program*: Instructions for the computer
  - Formal description in programming language
    - Statements (steps, actions)
    - Control structures (flow of control)

Structured Programming

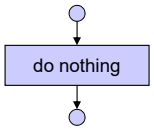
- Control Flow Chart
  - Graphical representation of program control flow
  - Example:



Structured Programming

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

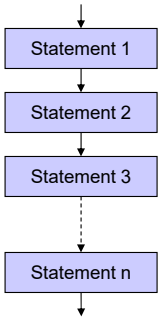
```
{
 /* nothing */
}
```



Structured Programming

- Sequential execution in C
  - Statement blocks: *Compound statements*
  - Sequence of statements grouped by braces: **{ }**
- Example: Flow chart:

```
{
  /* statement 1 */

  /* statement 2 */

  /* statement 3 */

  /* ... */

  /* statement n */
}
```



Structured Programming

- Sequential execution in C
  - Statement blocks: *Compound statements*
  - 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... */
```

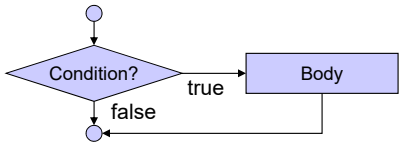

Structured Programming

- Sequential execution in C
  - Statement blocks: *Compound statements*
  - 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) {                                indentation level 0
    printf("%d is negative!", x);           indentation level 1
    /* handle negative values of x... */    indentation level 1
    if (x < -100) {                         indentation level 1
        printf("%d is too small!", x);      indentation level 2
        /* handle the problem... */         indentation level 2
        } /* fi */                          indentation level 2
    } /* fi */                              indentation level 1
if (x > 0) {                                indentation level 0
    printf("%d is positive!", x);           indentation level 1
    /* handle positive values of x... */    indentation level 1
    } /* fi */                              indentation level 1
/* more statements... */                    indentation level 0
```

Structured Programming

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



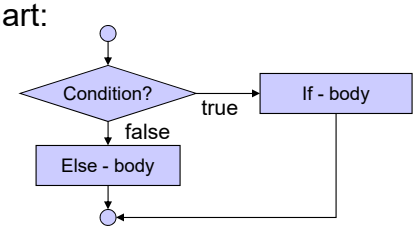
  - Example:

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

EECS10: Computational Methods in ECE, Lecture 7 (c) 2019 R. Doemer 17

Structured Programming

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



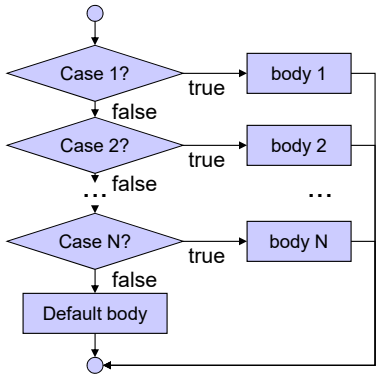
  - Example:

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

EECS10: Computational Methods in ECE, Lecture 7 (c) 2019 R. Doemer 18

Structured Programming

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

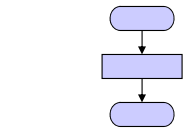


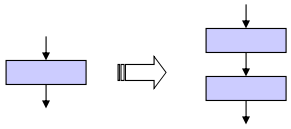
Example:

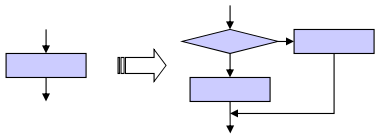
```
switch(LetterGrade)
{ case 'A':
   { printf("Excellent!");
     break; }
  case 'B':
  case 'C':
  case 'D':
   { printf("Passed.");
     break; }
  case 'F':
   { printf("Failed!");
     break; }
  default:
   { printf("Invalid grade!");
     break; }
} /* hctiws */
```


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

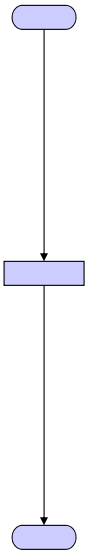






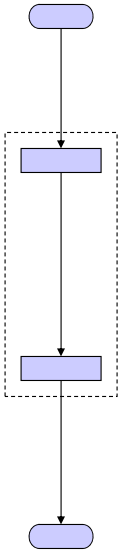
Structured Program Composition

- Example:
  - Initial flow chart



Structured Program Composition

- Example:
  - Sequential composition

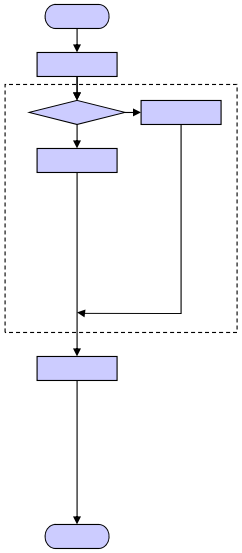


Structured Program Composition

- Example:
  - insertion of another sequential statement

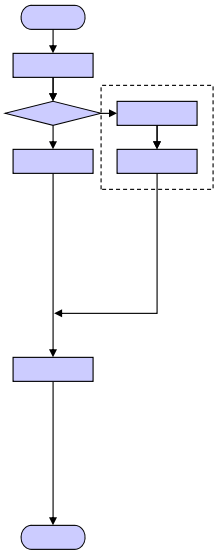
Structured Program Composition

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



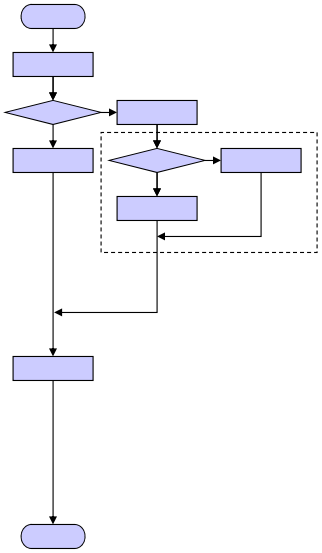
Structured Program Composition

- Example:
  - insertion of sequential statement



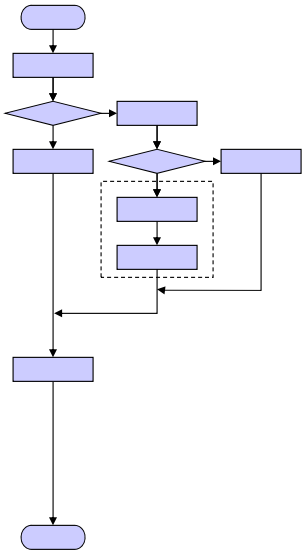
Structured Program Composition

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



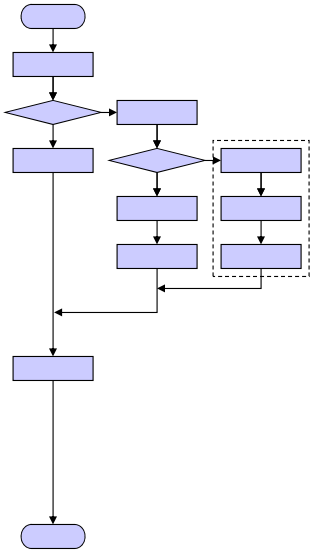
Structured Program Composition

- Example:
  - insertion of sequential statement



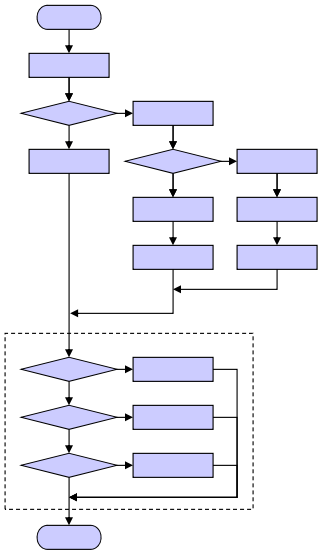
Structured Program Composition

- Example:
  - insertion of sequential statement (twice)



Structured Program Composition

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



Example Program

- Grade calculation: **`Grade.c`** (part 1/3)

```
/* Grade.c: convert score into letter grade     */
/* author: Rainer Doemer                        */
/* modifications:                               */
/* 10/17/04 RD  initial version                 */

#include <stdio.h>

/* main function */

int main(void)
{
   /* variable definitions */
   int  score = 0;
   char grade;

   /* input section */
   while (score < 1 || score > 100)
      { printf("Please enter your score (1-100): ");
        scanf("%d", &score);
       } /* elihw */
...
```

Example Program

- Grade calculation: **`Grade.c`** (part 2/3)

```
...
   /* computation section */
   if (score >= 90)
      { grade = 'A'; }
   else
      { if (score >= 80)
           { grade = 'B'; }
        else
           { if (score >= 70)
                { grade = 'C'; }
             else
                { if (score >= 60)
                     { grade = 'D'; }
                  else
                     { grade = 'F'; }
                } /* esle */
           } /* esle */
      } /* esle */
...
```


Example Program

- Grade calculation: **`Grade.c`** (part 3/3)

```
...

   /* output section */
   printf("Your letter grade is %c.\n", grade);

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

/* EOF */
```

Example Program

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

```
% vi Grade.c
% gcc Grade.c -o Grade -Wall -ansi
% ./Grade
Please enter your score (1-100): 111
Please enter your score (1-100): 99
Your letter grade is A.
% Grade
Please enter your score (1-100): 85
Your letter grade is B.
% Grade
Please enter your score (1-100): 71
Your letter grade is C.
% Grade
Please enter your score (1-100): 69
Your letter grade is D.
% Grade
Please enter your score (1-100): 55
Your letter grade is F.
%
```


Example Program

- Grade calculation: **`Grade2.c`** (part 1/3)

```
/* Grade2.c: convert score into letter grade    */
/* author: Rainer Doemer                        */
/* modifications:                               */
/* 10/18/04 RD  use 'switch' statement          */
/* 10/17/04 RD  initial version                 */

#include <stdio.h>

/* main function */

int main(void)
{
   /* variable definitions */
   int  score = 0;
   char grade;

   /* input section */
   while (score < 1 || score > 100)
      { printf("Please enter your score (1-100): ");
        scanf("%d", &score);
       } /* elihw */
...
```

Example Program

- Grade calculation: **`Grade2.c`** (part 2/3)

```
.../* computation section */
    switch (score / 10)
      { case 10:
        case 9:
           { grade = 'A';
             break; }
        case 8:
           { grade = 'B';
             break; }
        case 7:
           { grade = 'C';
             break; }
        case 6:
           { grade = 'D';
             break; }
        default:
           { grade = 'F';
             break; }
       } /* hctiws */
...
```


Example Program

- Grade calculation: **`Grade2.c`** (part 3/3)

```
...

   /* output section */
   printf("Your letter grade is %c.\n", grade);

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

/* EOF */
```

Example Program

- Example session: **`Grade2.c`**

```
% cp Grade.c Grade2.c
% vi Grade2.c
% gcc Grade2.c -o Grade2 -Wall -ansi
% ./Grade2
Please enter your score (1-100): 111
Please enter your score (1-100): 99
Your letter grade is A.
% Grade2
Please enter your score (1-100): 85
Your letter grade is B.
% Grade2
Please enter your score (1-100): 71
Your letter grade is C.
% Grade2
Please enter your score (1-100): 69
Your letter grade is D.
% Grade2
Please enter your score (1-100): 55
Your letter grade is F.
%
```