

EECS 22: Advanced C Programming

Lecture 4

Rainer Dömer

doemer@uci.edu

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

Lecture 4: 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

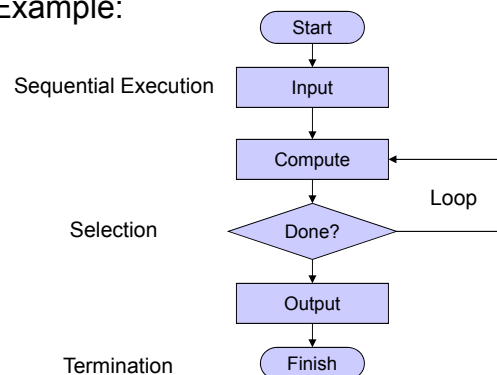
EECS22: Advanced C Programming, Lecture 4

(c) 2016 R. Doemer

3

Structured Programming

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



EECS22: Advanced C Programming, Lecture 4

(c) 2016 R. Doemer

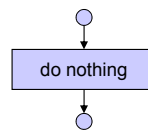
4

Structured Programming

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

Flow chart:

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



EECS22: Advanced C Programming, Lecture 4

(c) 2016 R. Doemer

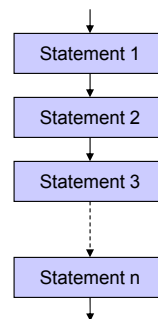
5

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 */
}
```



EECS22: Advanced C Programming, Lecture 4

(c) 2016 R. Doemer

6

Structured Programming

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

```

{ /* declarations */
  /* definitions */

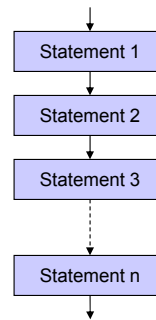
  /* statement 1 */
  /* statement 2 */
  /* statement 3 */

  /* ... */

  /* statement n */
}

```

Flow chart:



EECS22: Advanced C Programming, Lecture 4

(c) 2016 R. Doemer

7

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... */

```

EECS22: Advanced C Programming, Lecture 4

(c) 2016 R. Doemer

8

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

```

EECS22: Advanced C Programming, Lecture 4

(c) 2016 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 4

(c) 2016 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... */
```

EECS22: Advanced C Programming, Lecture 4

(c) 2016 R. Doemer

11

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... */
```

EECS22: Advanced C Programming, Lecture 4

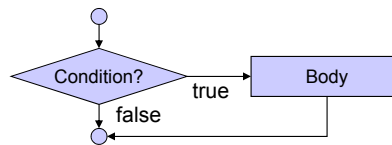
(c) 2016 R. Doemer

12

Structured Programming

- Selection: **if** statement

- Flow chart:



- Example:

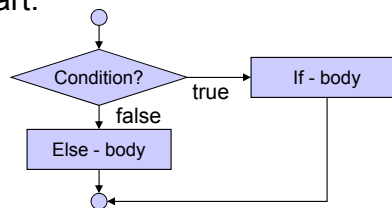
```

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

Structured Programming

- Selection: **if-else** statement

- Flow chart:



- Example:

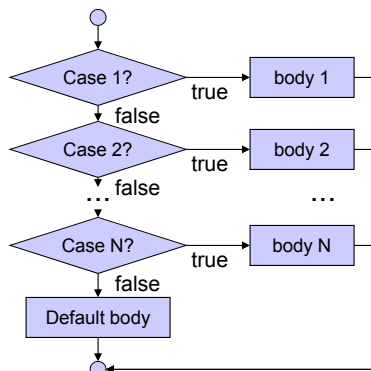
```

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 4

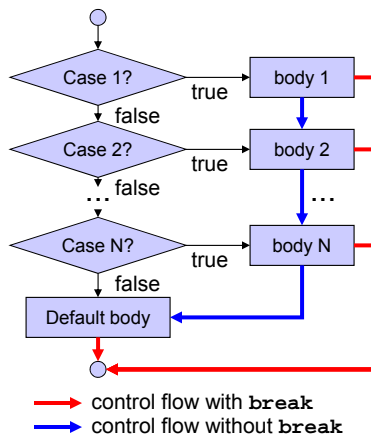
(c) 2016 R. Doemer

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 4

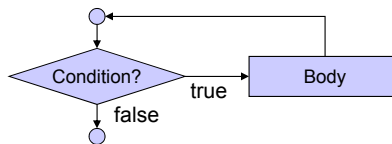
(c) 2016 R. Doemer

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!

EECS22: Advanced C Programming, Lecture 4

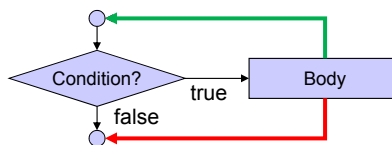
(c) 2016 R. Doemer

17

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!

EECS22: Advanced C Programming, Lecture 4

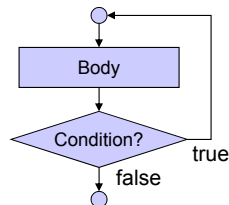
(c) 2016 R. Doemer

18

Structured Programming

- Repetition: **do-while** loop

– Flow chart:



– Example:

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

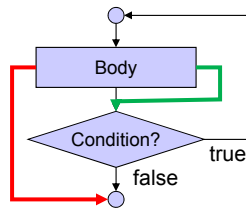
– Note:

- 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**

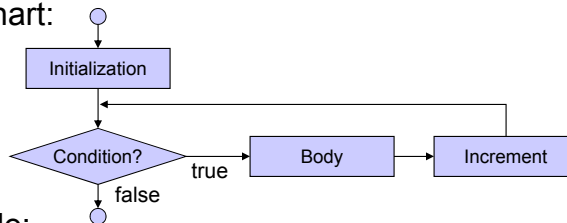
– Note:

- 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 }`

EECS22: Advanced C Programming, Lecture 4

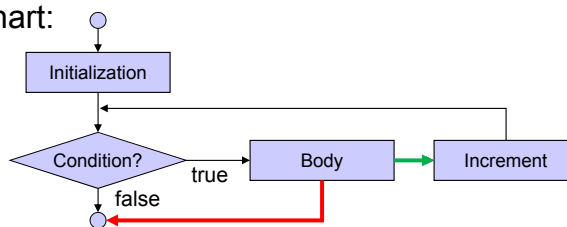
(c) 2016 R. Doemer

21

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 }`

EECS22: Advanced C Programming, Lecture 4

(c) 2016 R. Doemer

22

Arbitrary Control Flow

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

– Example:

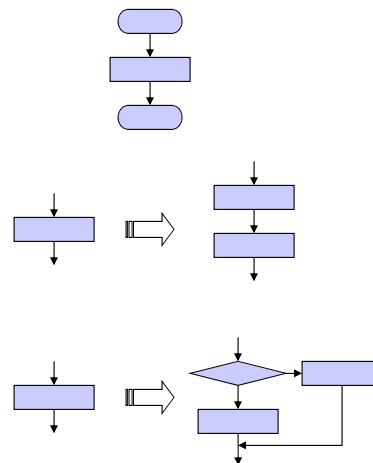
```
begin: count = 0;
      goto next;
repeat: if (count > 100)
        { goto end; }
next:  count++;
      if (count == 77)
        { goto next; }
      goto repeat;
end:   printf("%d", count);
```

– 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

```

    graph TD
      Start([Start]) --> Process[Process]
      Process --> End([End])
    
```

EECS22: Advanced C Programming, Lecture 4 (c) 2016 R. Doemer 25

Structured Program Composition

- Example:
 - Sequential composition

```

    graph TD
      Start([Start]) --> P1[Process 1]
      P1 --> P2[Process 2]
      P2 --> End([End])
      subgraph SequentialComposition [ ]
        P1
        P2
      end
    
```

EECS22: Advanced C Programming, Lecture 4 (c) 2016 R. Doemer 26

Structured Program Composition

- Example:
 - insertion of another sequential statement

The flowchart illustrates a sequential process. It starts with an oval representing the start of the program. An arrow points down to a dashed rectangular box containing two rectangular process blocks stacked vertically. An arrow points down from the bottom of this dashed box to a single rectangular process block. Finally, an arrow points down to an oval representing the end of the program.

EECS22: Advanced C Programming, Lecture 4
(c) 2016 R. Doemer
27

Structured Program Composition

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

The flowchart illustrates a program structure with an if-else statement. It starts with an oval representing the start of the program. An arrow points down to a rectangular process block. Another arrow points down to a dashed rectangular box containing a diamond-shaped decision block. From the right side of the diamond, an arrow points to a rectangular process block. From the bottom of the diamond, an arrow points down to another rectangular process block. An arrow from the right-side process block loops back to the bottom of the diamond. Finally, an arrow points down from the bottom process block to an oval representing the end of the program.

EECS22: Advanced C Programming, Lecture 4
(c) 2016 R. Doemer
28

Structured Program Composition

- Example:
 - insertion of sequential statement

The flowchart shows a sequence of operations: a start node, a process box, a decision diamond, another process box, and a final end node. A dashed box highlights a new process box and a second process box being inserted between the first process box and the decision diamond. Arrows indicate the flow from the start node to the first process box, then to the decision diamond, then to the first process box, then to the second process box, then to the new process box, and finally to the end node.

EECS22: Advanced C Programming, Lecture 4 (c) 2016 R. Doemer 29

Structured Program Composition

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

The flowchart shows a sequence of operations: a start node, a process box, a decision diamond, another process box, and a final end node. A dashed box highlights a new decision diamond and two process boxes being inserted between the first process box and the decision diamond. Arrows indicate the flow from the start node to the first process box, then to the decision diamond, then to the first process box, then to the new decision diamond, then to the first process box, then to the second process box, then to the new decision diamond, and finally to the end node.

EECS22: Advanced C Programming, Lecture 4 (c) 2016 R. Doemer 30

Structured Program Composition

- Example:
 - insertion of sequential statement

The flowchart shows a sequence of operations starting from a start node (oval). It proceeds through a series of rectangular process boxes. A diamond-shaped decision node is reached. One path from the diamond leads to a process box, then to another diamond. From this second diamond, one path leads to a process box and then to a dashed rectangular box containing two sequential process boxes. The other path from the second diamond loops back to the process box immediately preceding it. After exiting the dashed box, the flow returns to the process box immediately following the first diamond. The sequence continues through another process box and ends at a final node (oval).

EECS22: Advanced C Programming, Lecture 4 (c) 2016 R. Doemer 31

Structured Program Composition

- Example:
 - insertion of sequential statement (twice)

The flowchart is similar to the one on slide 31, but the dashed rectangular box contains three sequential process boxes instead of two. The flow starts at a start node, goes through a process box to a diamond. One path leads to a process box, then to a second diamond. From the second diamond, one path leads to a process box and then to the dashed box containing three sequential process boxes. The other path from the second diamond loops back to the process box immediately preceding it. After exiting the dashed box, the flow returns to the process box immediately following the first diamond. The sequence continues through another process box and ends at a final node.

EECS22: Advanced C Programming, Lecture 4 (c) 2016 R. Doemer 32

Structured Program Composition

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

EECS22: Advanced C Programming, Lecture 4 (c) 2016 R. Doemer 33

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!

EECS22: Advanced C Programming, Lecture 4 (c) 2016 R. Doemer 34

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 4

(c) 2016 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 4

(c) 2016 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.  
%
```