Computer Programming using C

Flow Control-Part I

Instructor: HOU, Fen

2025

Categories of Flow Controls

- *Relational*, *Equality*, *Logical* Operators are provided to facilitate flow controls.
- *Sequential* – Statements in a program are executed one after another.
- *Selection* – A choice of alternative actions (**`if`**, **`if-else`**, and **`switch`**).
- *Repetition* – Repeat certain actions (**`while`**, **`for`**, and **`do-while`**).

1. Relational Operators and Expressions

- Relational Operators

  < > <= >=

  - All relational operators are binary operators, taking two operands.

- Relational Expressions
  - A *relational expression* is an expression that involves the use of one or more relational operators.
  - Relational expressions yield either *true* or *false*.
  - "True" is represented by integer 1.
  - "False" is represented by integer 0.

Examples

Relational Expressions	Expression Values	
7 < 9	true	1
if **a** is **5**, **`a > 3`**		
if **a** is **2**, **`a > 3`**		
if **b** is **40** and c is 6, **`b <= (9 * 3 + c)`**		

Examples

Relational Expressions	Expression Values	
7 < 9	true	1
if **a** is **5**, **`a > 3`**	true	1
if **a** is **2**, **`a > 3`**	false	0
if **b** is **40** and c is 6, **`b <= (9 * 3 + c)`**	false	0

Operator precedence and associativity

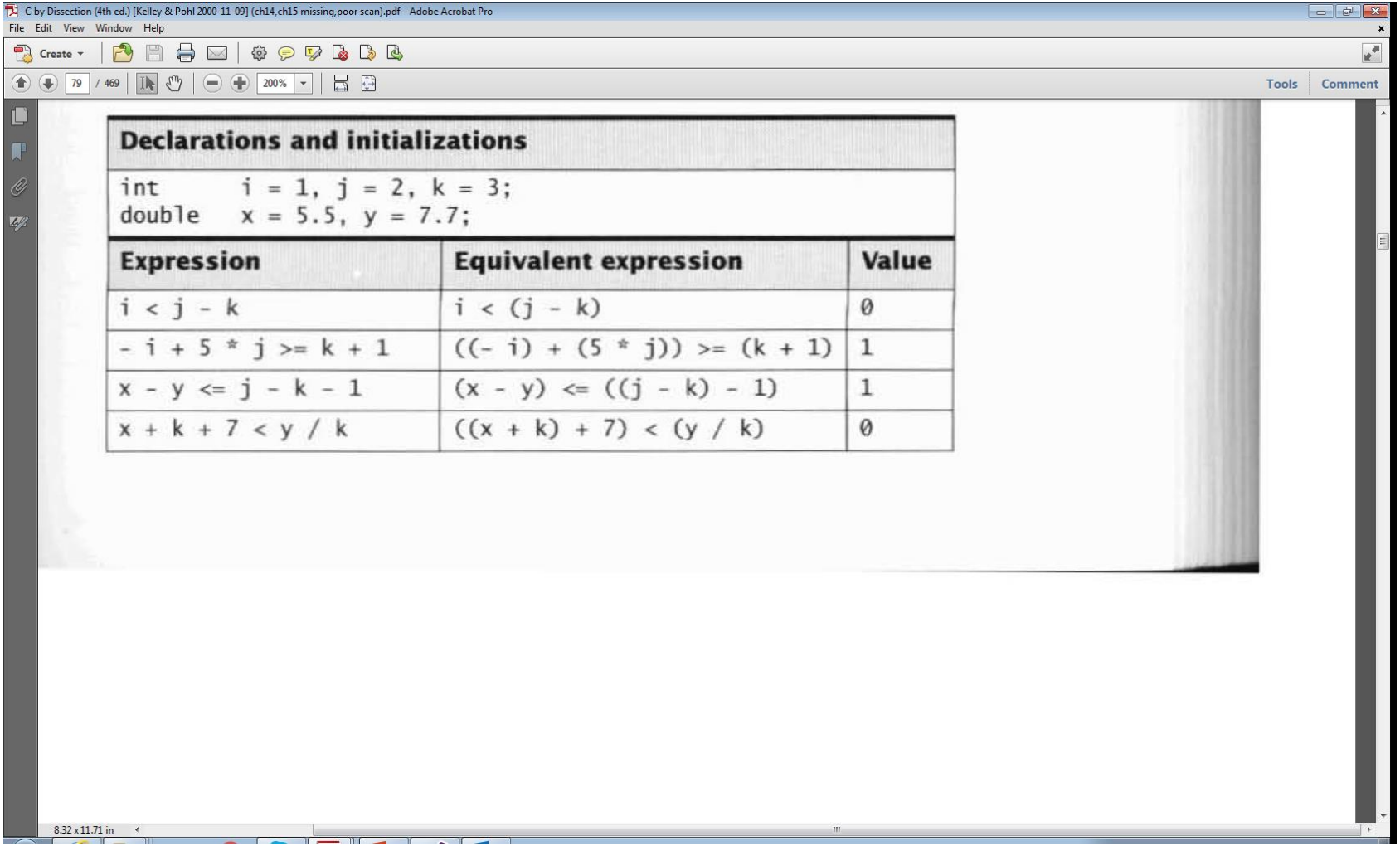
Operator	Associativity	Precedence
( ) ++ (postfix) -- (postfix)	left to right	Highest
+ (unary) - (unary) ++ (prefix) -- (prefix) !	right to left	
* / %	left to right	
+ -	left to right	
< <= > >=	left to right	
== !=	left to right	
&&	left to right	
||	left to right	
?:	right to left	
= += -= *= /= etc.	right to left	
, (comma operator)	left to right	Lowest

Exercise

Declarations and Initializations
`int i=1, j=2, k=3;` `double x=5.5, y=7.7;`

Expression	Equivalent Expression	Value
`i < j - k`	`i < (j - k)`	0
`- i + 5 * j >= k + 1`		
`x - y <= j - k - 1`		
`x + k + 7 < y / k`		

Exercise



Declarations and initializations		
int i = 1, j = 2, k = 3; double x = 5.5, y = 7.7;		
Expression	**Equivalent expression**	**Value**
i < j - k	i < (j - k)	0
- i + 5 * j >= k + 1	((- i) + (5 * j)) >= (k + 1)	1
x - y <= j - k - 1	(x - y) <= ((j - k) - 1)	1
x + k + 7 < y / k	((x + k) + 7) < (y / k)	0

2. Equality Operators and Expressions

== (Equal-to) **!=** (Not-Equal-to)

- The *equality operators* are binary operators, taking two operands.
- An *equality expression* yields either 1 (true) or 0 (false).
- Examples,

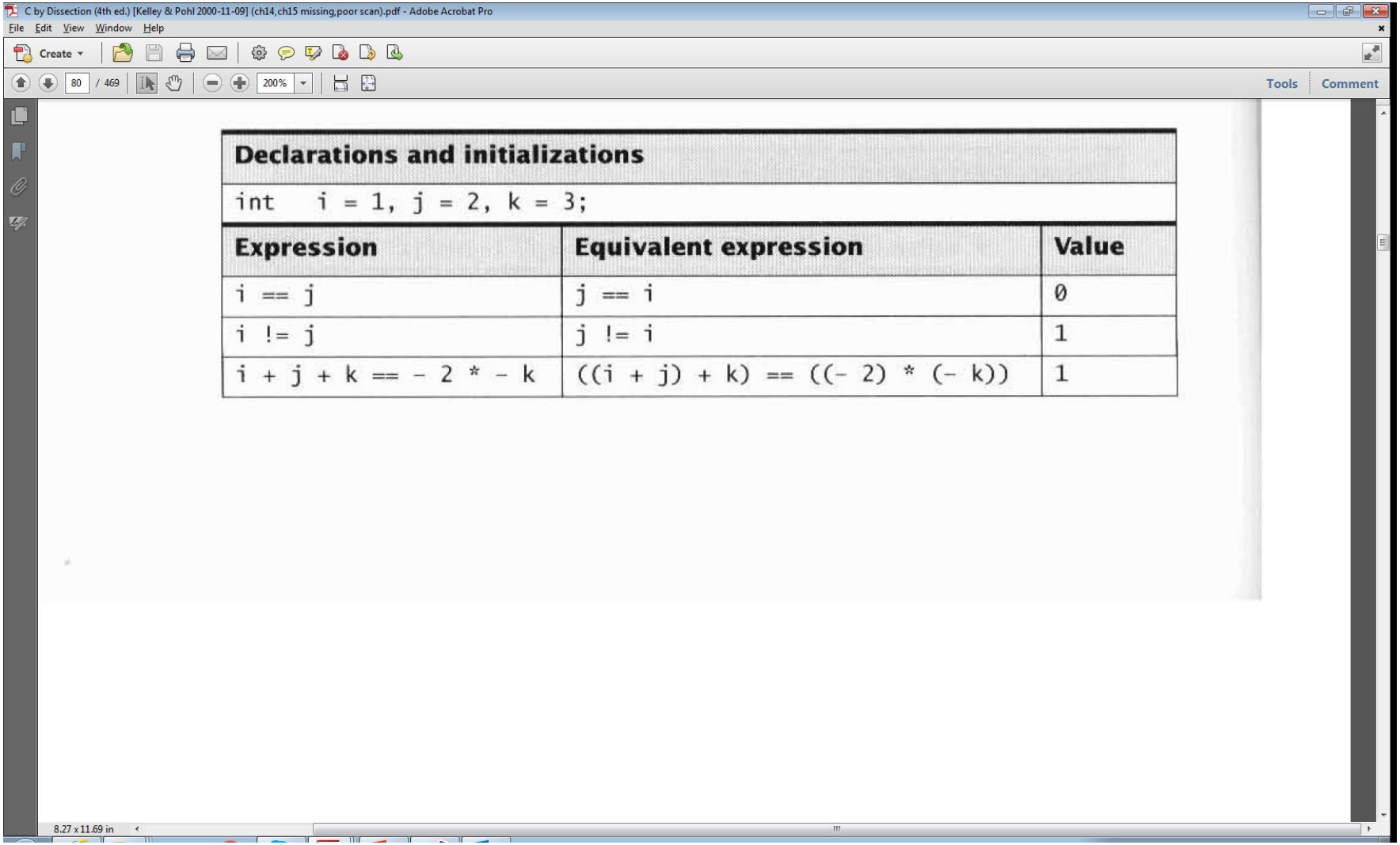
```
9 == 10
c == 'A'
'A' != 'B'
k != -2
area != (length * length)
```

Exercise

Declarations and Initializations
`int i=1, j=2, k=3;`

Expression	Equivalent Expression	Value
`i == j`	`j == i`	0
`i != j`		
`i + j + k == - 2 * - k`		

Equality Operators and Expressions



Declarations and initializations		
`int i = 1, j = 2, k = 3;`		
Expression	**Equivalent expression**	**Value**
`i == j`	`j == i`	`0`
`i != j`	`j != i`	`1`
`i + j + k == - 2 * - k`	`((i + j) + k) == ((- 2) * (- k))`	`1`

3. Logical Operators and Expressions

! (Not) **&&** (And) **||** (Or)

- Operator **!** is unary, taking only one operand.
- Operators **&&** and **||** are binary, taking two operands.
- Logical expressions yield either 1 (true) or 0 (false).

Logical Operators: AND (&&) , OR (||) and NOT (!)

- Operator AND (&&)
  - Meaning: Logical AND. Ture only if all operands are true
  - Example: if c=5 and d=2, expression ((c==5) && (d>5)) is false.
- Operator OR (||)
  - Meaning: Logical OR. True only if either one operand is true.
  - Example: if c=5 and d=2, expression ((c==5) && (d>5)) is true
- Operator NOT (!)
  - Meaning: Logical NOT. Ture only if the operand is false (or 0).
  - Example: if c=5 and d=2, expression ! (c==5) is false

Expression as an Operand to Logical Expressions

- Note carefully that when an expression is used as an operand to a logical expression,
  - if the operand expression has the value *zero*, it will be regarded as *false* when evaluating the logical expression.
  - if the operand expression has a *non-zero* value, it will be regarded as *true* when evaluating the logical expression.

Logical Operators: AND (&&) and OR (||)

- The binary logical operators && and || also act on expressions and yield either the integer value 0 or the integer value 1.
- Correct logical expressions:
  - !(a<b) && c
  - 3 && (-2 *a +7)
- Incorrect logical expressions:
  - a && /*one operand missing*/
  - a | | b /*extra space not allowed */
  - a & b /* this is a bitwise AND operation */
  - &b /* the address of b */

3.1 The NOT (`!`) Operator

NOT Expression	Value
`!9`	
`!0`	
if **a** is **3**, `!a`	
if **x** is **2**, `!(x + 7.7)`	
if **a < b** is false and **c < d** is true, `!(a < b || c < d)`	

- What is the value of the logical expression below?

```
!!5
```

- And what will be printed by the following statement?

```
printf("%d", !!5);
```

3.1 The NOT (`!`) Operator

NOT Expression	Value
`!9`	0
`!0`	1
if **a** is **3**, `!a`	0
if **x** is **2**, `!(x + 7.7)`	0
if **a < b** is false and **c < d** is true, `!(a < b || c < d)`	0

- What is the value of the logical expression below?

```
!!5
```

- And what will be printed by the following statement?

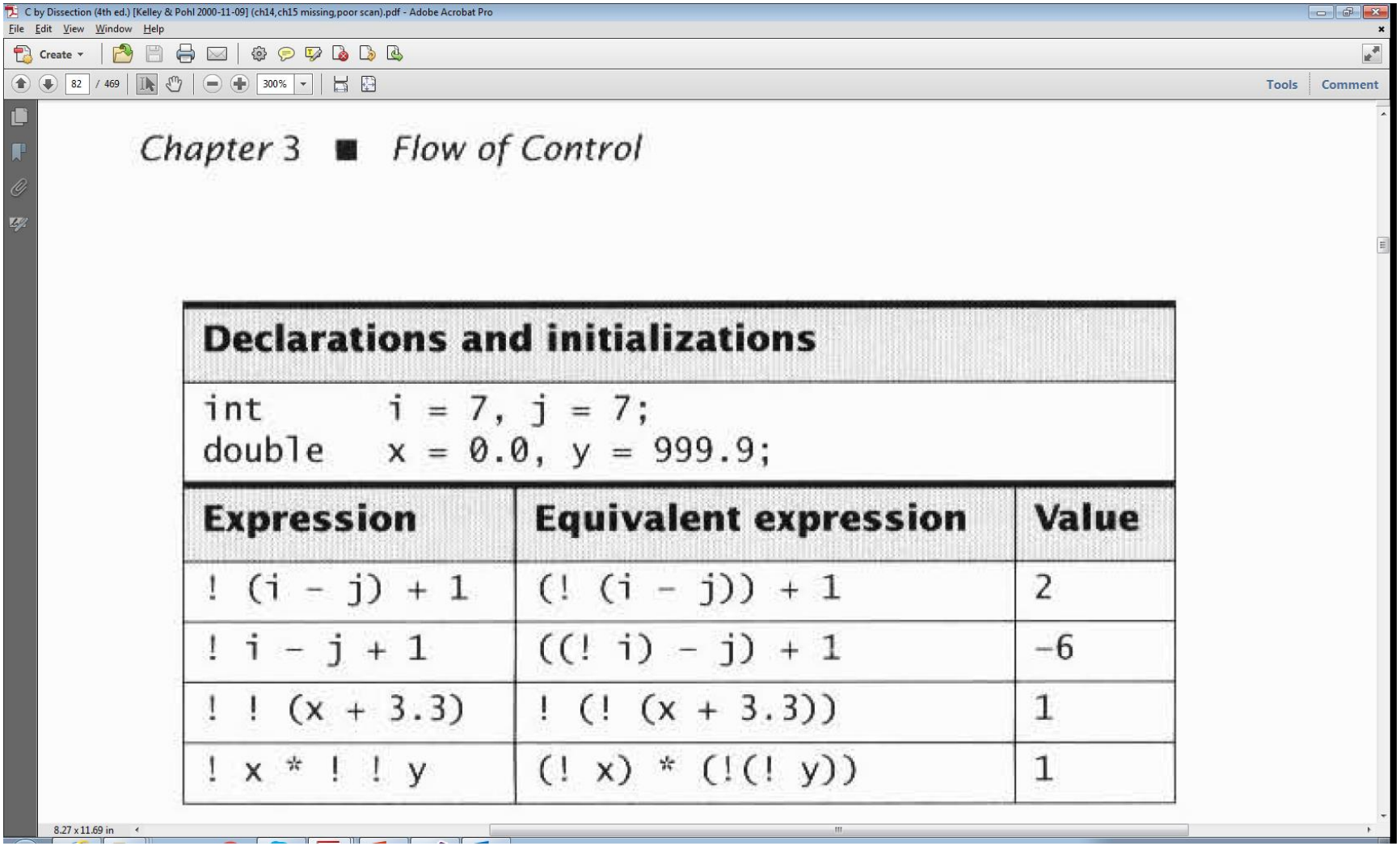
```
printf("%d", !!5);
```

Exercise

Declarations and Initializations
`int i=7, j=7;` `double x=0.0, y=999.9;`

Expression	Equivalent Expression	Value
`! (i - j) + 1`	`(! (i - j)) + 1`	2
`! i - j + 1`		
`! ! (x + 3.3)`		
`! x * ! ! y`		

Exercise



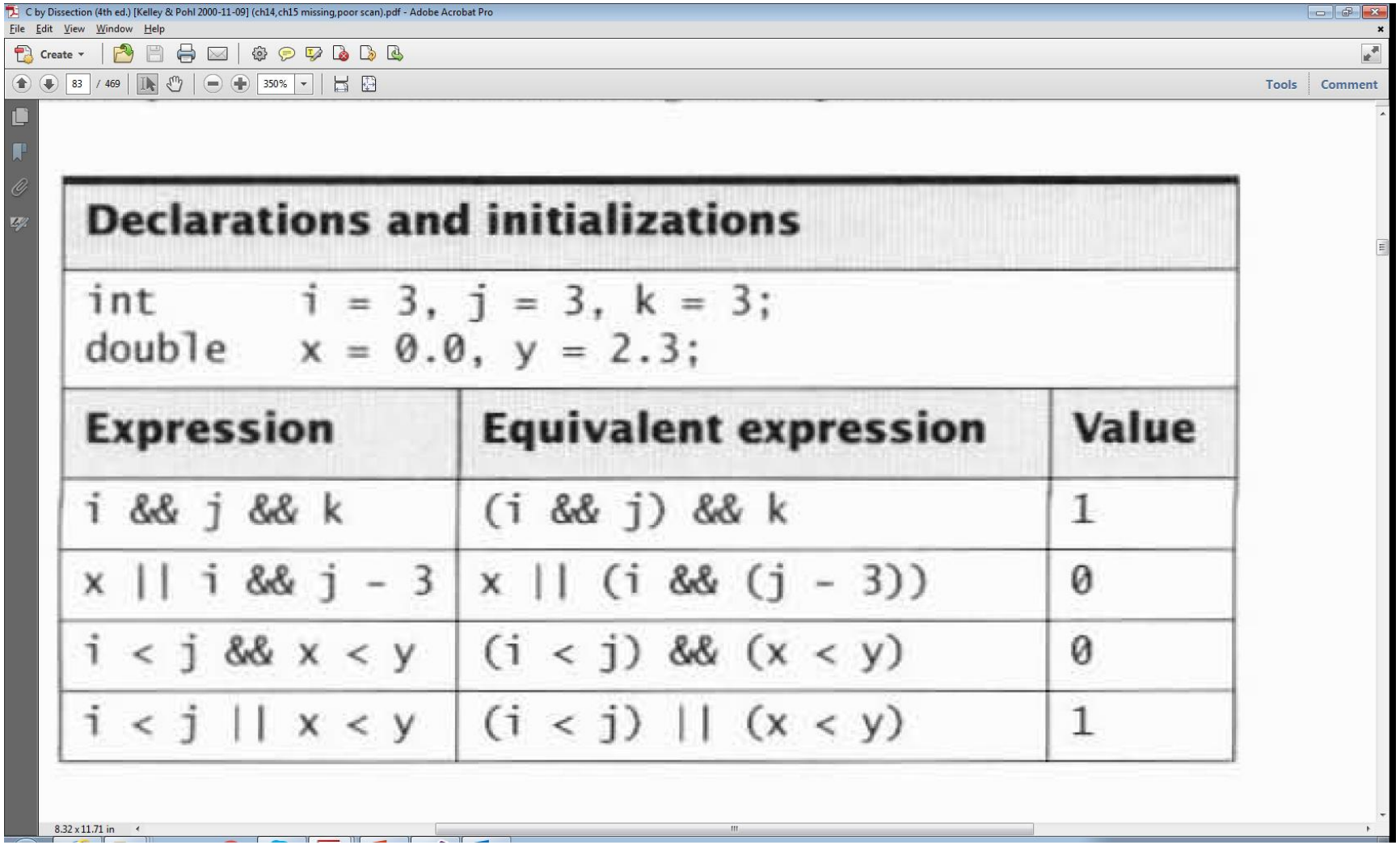
Declarations and initializations		
`int i = 7, j = 7;` `double x = 0.0, y = 999.9;`		
Expression	**Equivalent expression**	**Value**
`! (i - j) + 1`	`(! (i - j)) + 1`	2
`! i - j + 1`	`((! i) - j) + 1`	-6
`! ! (x + 3.3)`	`! (! (x + 3.3))`	1
`! x * ! ! y`	`(! x) * (!(! y))`	1

3.2 The AND (&&) and OR (||) Operators

Declarations and Initializations
`int i=3, j=3, k=3;` `double x=0.0, y=2.3;`

Expression	Equivalent Expression	Value
`i && j && k`	`(i && j) && k`	1
`x || i && j - 3`		
`i < j && x < y`		
`i < j || x < y`		

Logical Operators: AND (&&) and OR (||)



Declarations and initializations

```
int     i = 3, j = 3, k = 3;
double  x = 0.0, y = 2.3;
```

Expression	Equivalent expression	Value
`i && j && k`	`(i && j) && k`	1
`x || i && j - 3`	`x || (i && (j - 3))`	0
`i < j && x < y`	`(i < j) && (x < y)`	0
`i < j || x < y`	`(i < j) || (x < y)`	1

4. Short-Circuit Evaluations

- Evaluation of expressions containing **&&** and **||** stops as soon as the outcome (true or false) is known.

- **`expr1 && expr2`**
  - If **`expr1`** evaluates to false (zero), the evaluation of **`expr2`** will *not* occur.

- **`expr1 || expr2`**
  - If **`expr1`** evaluates to true (non-zero), the evaluation of **`expr2`** will *not* occur.

Short-Circuit examples

```
1  int i, j;
2
3  i=2 && (j=2);
4  printf ("%d %d\n", i, j);      /* ? ? is printed */
5
6  (i=0) && (j=3);
7  printf ("%d %d\n", i, j);      /* ? ? is printed */
8
9  i=0 || (j=4);
10 printf ("%d %d\n", i, j);      /* ? ? is printed */
11
12 (i=2) || (j=5);
13 printf ("%d %d\n", i, j);      /* ? ? is printed */
```

Note carefully the effect of parenthesis in the above expressions! Refer to the operator precedence table if needed.

Short-Circuit examples

```
1  int i, j;
2
3  i=2 && (j=2);
4  printf ("%d %d\n", i, j);      /* 1 2 is printed */
5
6  (i=0) && (j=3);
7  printf ("%d %d\n", i, j);      /* 0 2 is printed */
8
9  i=0 || (j=4);
10 printf ("%d %d\n", i, j);      /* 1 4 is printed */
11
12 (i=2) || (j=5);
13 printf ("%d %d\n", i, j);      /* 2 4 is printed */
```

Note carefully the effect of parenthesis in the above expressions! Refer to the operator precedence table if needed.

Relational, Equality, and Logical Operators

Relational operators	less than	<
	greater than	>
	less than or equal to	<=
	greater than or equal to	>=
Equality operators	equal to	==
	not equal to	!=
Logical operators	(unary) negation	!
	logical and	&&
	logical or	||

5. The Compound Statement

- A series of variable declarations and statements surrounded by a pair of braces **{ }**.
- The group of declarations and statements will then be regarded as a single logical unit by the compiler.

Example

```
int main(void)
{
    int x, y;
    scanf ("%d", &x);
    scanf ("%d", &y);
    {
        int temp;
        temp = x;
        x = y;
        y = temp;
    }
}
```

Another Example

- ```
  if (...)
      one_statement;
  ```

- ```
  if (...) {
      one_statement;
      another_statement;
  }
  ```

6. Selection – The `if` Statement

```
if (expr)
    statement;
next_statement;
```

- If **`expr`** is true (nonzero), then **`statement`** is executed.
- Otherwise **`statement`** is skipped, and control is passed to **`next_statement`**.

Examples

```
1  score = 80;
2  if (score >= 50)
3      printf ("Congratulations!\n");
4  printf ("Your score is %d.\n", score);
```

```
Congratulations!
Your score is 80.
```

```
1  score = 40;
2  if (score >= 50)
3      printf ("Congratulations!\n");
4  printf ("Your score is %d.\n", score);
```

```
Your score is 40.
```

`if` and Compound Statements

```
if (score >= 50) {
    grade = 'P';
    printf ("Congratulations!\n");
}
printf ("Your score is %d\n.", score);
```

7. Selection – The **`if-else`** Statement

```
if (expr)
    statement_1;
else
    statement_2;
next_statement;
```

- If **`expr`** is true (nonzero), then **`statement_1`** is executed.
- Otherwise **`statement_1`** is skipped and **`statement_2`** is executed.
- In both cases, control is then passed to **`next_statement`**.

Examples

```
1  if (score >= 50) {
2      grade = 'P';
3      printf ("Congratulations!\n");
4  } else
5      grade = 'F';
6  printf ("Your score is %d.\n", score);
7  printf ("Your grade is %c.\n", grade);
```

When score is 80

When score is 40

Examples

```
1  if (score >= 50) {
2      grade = 'P';
3      printf ("Congratulations!\n");
4  } else
5      grade = 'F';
6  printf ("Your score is %d.\n", score);
7  printf ("Your grade is %c.\n", grade);
```

When score is 80

```
Congratulations!
Your score is 80.
Your grade is P.
```

When score is 40

```
Your score is 40.
Your grade is F.
```

7.1 The "Dangling **`else`**" Problem

- An **`if`** statement can be used as the statement part of another if statement.

```
if (score >= 50)
    if (score >= 85)
        printf ("Congratulations!\n");
printf ("Your score is %d.\n", score);
```

- What will be printed by the above program fragment if score is
  (a) 40 ?
  (b) 70 ?
  (c) 90 ?

The "Dangling **`else`**" Problem

- An **`if-else`** statement can also be used as the statement part of another if statement.

- However, there are two possible interpretations:
- **Case I**

```
if (score >= 50)
    if (score >= 85)
        printf ("Congratulations!\n");
    else
        printf ("You got a pass.\n");
```

- **Case II**

```
if (score >= 50)
    if (score >= 85)
        printf ("Congratulations!\n");
else
    printf ("You got a pass.\n");
```

`if-else` Pairing Rule

- An **`else`** statement attaches to the nearest **`if`** that has not been paired with an **`else`**.

- So Case I has a better *indentation* for semantics of the program fragment.

```c
#include <stdio.h>

int main(void)
{
    int x, y, z, min;

    printf("Input three integers: ");
    scanf("%d%d%d", &x, &y, &z);

    if (x < y)
        min = x;
    else
        min = y;

    if (z < min)
        min = z;

    printf("The minimum value is %d.\n", min);

    return (0);
}
```

Example

This program finds and prints the minimum among the three values input by the user.

7.2 Deeply Nested if-else Statements (Different styles to express)

```
if (score >= 85)
    grade = 'A';
else
    if (score >= 80)
        grade = 'B';
    else
        if (score >= 70)
            grade = 'C';
        else
            if (score >= 60)
                grade = 'D';
            else
                if (score >= 50)
                    grade = 'E';
                else
                    grade = 'F';
```

```
if (score >= 85)
    grade = 'A';
else
if (score >= 80)
    grade = 'B';
else
if (score >= 70)
    grade = 'C';
else
if (score >= 60)
    grade = 'D';
else
if (score >= 50)
    grade = 'E';
else
    grade = 'F';
```

8. Selection – The **`switch`** Statement

Syntax and Semantics

```
switch (expression) {
    case constant_expr_1:
        statement_1;
        break;
    case constant_expr_2:
        statement_2;
        break;
    ....
    case constant_expr_n:
        statement_n;
        break;
    default:
        statement_d;
}
next_statement;
```

8. Selection – The **switch** Statement

Syntax and Semantics

```
switch (expression) {
    case constant_expr_1:
        statement_1;
        break;
    case constant_expr_2:
        statement_2;
        break;
    ....
    case constant_expr_n:
        statement_n;
        break;
    default:
        statement_d;
}
next_statement;
```

1. Evaluate **expression**, whose result must be a simple data type (such as **int**, **char**).

2. Compare the result with each **constant_expr_*i*** in turn until a matching is found.

8. Selection – The **switch** Statement

Syntax and Semantics

```
switch (expression) {
    case constant_expr_1:
        statement_1;
        break;
    case constant_expr_2:
        statement_2;
        break;
    ....
    case constant_expr_n:
        statement_n;
        break;
    default:
        statement_d;
}
next_statement;
```

3. Say **constant_expr_2** is matched.
   - Execute **statement_2**, **statement_3**, …, **statement_n**, **statement_d**.
   - However, whenever a break is reached, jump to **next_statement** immediately.

8. Selection – The **`switch`** Statement

Syntax and Semantics

```
switch (expression) {
    case constant_expr_1:
        statement_1;
        break;
    case constant_expr_2:
        statement_2;
        break;
    ....
    case constant_expr_n:
        statement_n;
        break;
    default:
        statement_d;
}
next_statement;
```

4. If no **`constant_expr_i`** matches, and **`default`** is present, execute **`statement_d`**.

5. Execute **`next_statement`**.

```
1  int n;
2  …
3  switch (n) {
4      case 1:
5          printf("One\n");
6          break;
7      case 2:
8          printf("Two\n");
9          break;
10     case 3:
11         printf("Three\n");
12         break;
13     case 4:
14         printf("Four\n");
15         break;
16     case 5:
17         printf("Five\n");
18         break;
19     default:
20         printf("Invalid number!\n");
21 }
```

If **n** is **2**

If **n** is **5**

If **n** is **10**

```
1  int n;
2  …
3  switch (n) {
4      case 1:
5          printf("One\n");
6          break;
7      case 2:
8          printf("Two\n");
9          break;
10     case 3:
11         printf("Three\n");
12         break;
13     case 4:
14         printf("Four\n");
15         break;
16     case 5:
17         printf("Five\n");
18         break;
19     default:
20         printf("Invalid number!\n");
21 }
```

If **n** is **2**

```
Two
```

If **n** is **5**

```
Five
```

If **n** is **10**

```
Invalid number!
```

```
1  int n;
2  …
3  switch (n) {
4      case 1:
5          printf("One\n");
6
7      case 2:
8          printf("Two\n");
9
10     case 3:
11         printf("Three\n");
12
13     case 4:
14         printf("Four\n");
15         break;
16     case 5:
17         printf("Five\n");
18
19     default:
20         printf("Invalid number!\n");
21 }
```

If **n** is **2**

If **n** is **5**

If **n** is **10**

```
1  int n;
2  …
3  switch (n) {
4      case 1:
5          printf("One\n");
6
7      case 2:
8          printf("Two\n");
9
10     case 3:
11         printf("Three\n");
12
13     case 4:
14         printf("Four\n");
15         break;
16     case 5:
17         printf("Five\n");
18
19     default:
20         printf("Invalid number!\n");
21 }
```

If **n** is **2**

```
Two
Three
Four
```

If **n** is **5**

```
Five
Invalid number!
```

If **n** is **10**

```
Invalid number!
```

9. Selection – The Conditional Operator **?:**

`expr1 ? expr2 : expr3`

- First, **`expr1`** is evaluated.
- If true, then **`expr2`** is evaluated and the value of **`expr2`** is taken as the value of the whole expression.
- Else, **`expr3`** is evaluated and the value of **`expr3`** is taken as the value of the whole expression.

Examples

```
printf("min=%d", (x < y) ? (min = x) : (min = y));

if (x < y)
    min = x;
else
    min = y;
printf("min=%d", min);



(y < z) ? printf("y is smaller") : printf("z is smaller");

if (y < z)
    printf("y is smaller");
else
    printf("z is smaller");
```