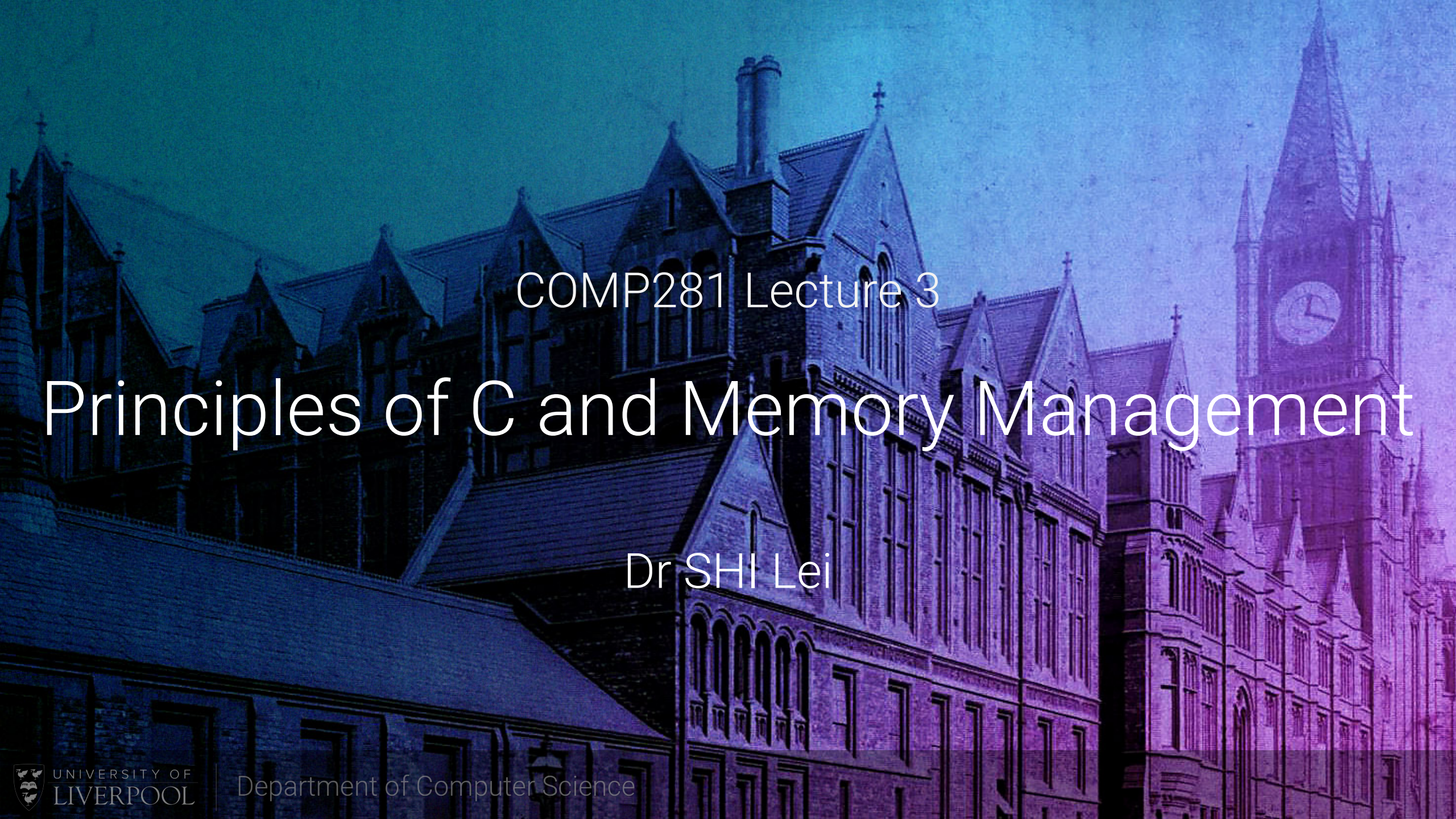


COMP281 Lecture 3

Principles of C and Memory Management

Dr SHI Lei

UNIVERSITY OF LIVERPOOL | Department of Computer Science

Last Lecture

- Compiling C Programs
  - 4 kinds of files to work with: **source code** files, **header** files, **object** files, and binary **executables**
  - The Preprocessor -> `#define`, `#include`
  - The Compiler -> `% gcc foo.c –o foo`
  - The Linker -> `% gcc bar.o baz.o -o myprogram`

Last Lecture

- C Language Basics
  - The `main()` function
  - C Program Skeleton -> segments
  - Identifiers -> naming variables, functions, etc.
  - Keywords -> reserved words, may not be used as identifier names
  - Basic Data Variables and Types
    - Integers, unsigned integers, floating point numbers, chars
  - Constants (fixed values) -> `const int x = 2;` `#define PI 3.14`

Today

- C Language Basics
  - Basic I/O
  - Operators
  - Decision Making

Basic I/O

The “Hello, world!” program

```c
#include <stdio.h>

int main(void)
{
    printf("Hello, World!\n");
    return 0;
}
```

The “Hello, world!” program

```
#include <stdio.h>

int main(void)

{

    printf("Hello, World!\n");

    return 0;

}
```

Basic I/O

- Output Function
  - `printf( )`
- Input Function
  - `scanf( )`

Output Function - `printf( )`

Output Function – `printf()`

- Send data to the standard output (usually the monitor) to be printed according to specific format.
- General format:

```
printf("Hi, there!");
printf("I have %d modules this term\n", sum);
```


Placeholder / format specifier

escape sequence

Output Function – printf()

```
printf(“I have %d modules this term\n”, sum);
```


Placeholder / format specifier

Starts with the character ‘%’;

Ends with one or two letters that indicate what "type" of formatting is necessary.

Between '%' and "type" field may appear "modifiers", "width", and "precision" fields.

```
% [flags] [minimum-field-width] [.precision] Type
```

where square brackets indicate that a field is optional.

Output Function – printf()

```
printf("I have %d modules this
```


Placeholder / format specifier

Starts with the character '%';

Ends with one or two letters that indicate

Between '%' and "type" field may appear "

```
% [flags] [minimum-field-width]
```

where square brackets indicate that a field is opti

How to print a '%' ?

Output Function – printf()

```
printf("I have %d modules this term\n", sum);
```


Placeholder / format specifier

```
% [flags] [minimum-field-width] [.precision] Type
```

The ***Type*** Field

Type	Used For	Example	Output
%c	a single character	`printf("%c", 'v')`	v
%s	a string	`printf("%s %s", ”Hi", "Lei")`	Hi, Lei
%d	a decimal integer	`printf("%d", 27)`	27
%o	an octal (base 8) integer	`printf("%o", 27)`	33
%x	a lowercase hexadecimal (base 16) integer	`printf("%x", 27)`	1b
%X	an uppercase hexadecimal (base 16) integer	`printf("%X", 27)`	1B
%f	a floating point number for floats	`printf("%f", 1.23)`	1.23
%e	a floating point number in scientific notation	`printf("%e", 3.14)`	1.23E+01
%%	**a % character**	`printf("%%")`	**%**

Output Function – printf()

```
printf("I have %d modules this term\n", sum);
```


Placeholder / format specifier

```
% [flags] [minimum-field-width] [.precision] Type
```

??

% [flags] [minimum-field-width] [.precision] Type

The flags field controls 'characters' that are added to a string. It can be zero or more (in any order) of:

Flag	Description	Example	Output
- (minus)	Left-align the output of this placeholder.	`printf("|%3i|%-3i|",12,12);`	|·12|12·|
+ (plus)	Prepends a plus for positive signed-numeric types. positive = +, negative = -.	`printf("%+i",17);`	+17
(space)	Prepends a space for positive signed-numeric types. positive = , negative = -.	`printf("|% i|",12);`	|·12|
0 (zero)	When the 'width' option is specified, prepends zeros for numeric types.	`printf("|%04i|",12);`	|0012|
# (hash)	For %#o (octal), include a leading zero. For %#x (hex), include a leading '0x'. For floating point conversion, include a decimal point even if no digits follow.	`printf("%#X",26);`	0X1A

% [flags] [minimum-field-width] [.precision] Type

Description	Example	Output
After converting any value to a string, the field width represents the minimum number of characters in the resulting string. If the converted value has fewer characters, then the resulting string is padded with spaces (or zeros) on the left (or right) by default (or if the appropriate flag is used.)	`printf("|%5s|","ABC");` `printf("|%-5s|","ABC");`	|ABC
ABC	
Sometimes the minimum field width isn't known at compile-time, and must be computed at run-time. (For example, printing a table where the width of a column depends on the widest column value in the input.) In this case the field width can be specified as an asterisk ("*"), which acts like a place-holder for an int value used for the field width. The value appears in the argument list before the expression being converted.	`printf("|%-*s|",5,"ABC");`	|ABC|

`% [flags] [minimum-field-width] [.precision] Type`

Description	Example	Output
For floating point numbers, it controls the number of digits printed after the decimal point	`printf("%.3f",3.1);`	3.100
If the number provided has more precision than is given, it will round	`printf("%.3f",3.1415);`	3.142
for integers, the precision it controls the minimum number of digits printed	`printf("%.3d",99);`	099
for strings, the precision controls the maximum length of the string displayed	`printf("%.3s\n","abcd" );`	abc

```
printf("I have %d modules this term\n", sum);
```


escape sequence

Escape sequence

`\a`	Beep
`\b`	Backspace
`\f`	Formfeed (for printing)
`\n`	Newline
`\r`	Carriage Return
`\t`	Horizontal Tab
`\v`	Vertical Tab
`\\`	Backslash
`\'`	Single quotation mark
`\"`	Double quotation mark
`\?`	Question mark

Input Function - `scanf( )`

Input Function – `scanf()`

- Read data from the standard input device (usually keyboard) and store it in a variable.
- General format:
  - `scanf("Format string", &variable);`
- Ampersand (&) operator:
  - C address of operator
  - It passes the **address** of the variable, not the variable itself
  - It tells the `scanf()` where to find the variable to store the new value

Input Function – scanf()

- Example:

```
int age;
printf("Enter your age: ");
scanf("%d", &age);
```

- Common Conversion Identifier used in `printf()` and `scanf()`

	printf	scanf
int	%d	%d
float	%f	%f
double	%f	%f
char	%c	%c
string	%s	%s

Operators

An operator is a symbol that tells the compiler to perform specific mathematical or logical functions.

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Bitwise Operators
- Assignment Operators
- Misc Operators

Arithmetic Operators

- Assume variable **A = 1** and variable **B = 2**, then −

Operator	Description	Example
+	Adds two operands.	A + B = 3
−	Subtracts second operand from the first.	A − B = -1
*	Multiplies both operands.	A * B = 2
/	Divides numerator by de-numerator.	B / A = 2
%	Modulus Operator and remainder of after an integer division.	B % A = 0
++	Increment operator increases the integer value by one.	A++ = 2
--	Decrement operator decreases the integer value by one.	A-- = 0

Relational Operators

- Assume variable **A = 1** and variable **B = 2**, then −

Operator	Description	Example
==	Checks if the values of two operands are **equal** or not. If yes, then the condition becomes true.	A == B is not true.
!=	Checks if the values of two operands are equal or not. If the values are **not equal**, then the condition becomes true.	A != B is true.
>	Checks if the value of left operand is **greater than** the value of right operand. If yes, then the condition becomes true.	A > B is not true.
<	Checks if the value of left operand is **less than** the value of right operand. If yes, then the condition becomes true.	A < B is true.
>=	Checks if the value of left operand is **greater than or equal** to the value of right operand. If yes, then the condition becomes true.	A >= B is not true.
<=	Checks if the value of left operand is **less than or equal** to the value of right operand. If yes, then the condition becomes true.	A <= B is true.

Logical Operators

- Assume variable **A = 1** and variable **B = 0**, then −

Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.	A && B is false.
||	Called Logical OR Operator. If any of the two operands is non-zero, then the condition becomes true.	A || B is true.
!	Called Logical NOT Operator. It is used to reverse the logical state of its operand. If a condition is true, then Logical NOT operator will make it false.	!(A && B) is true.

Bitwise Operators

- Bitwise operator works on bits and perform bit-by-bit operation. The truth tables for &, |, and ^ is as follows −

p	q	p & q	p | q	p ^ q
0	0	0	0	0
0	1	0	1	1
1	1	1	1	0
1	0	0	1	1

Assume **A = 60** and **B = 13** in binary format, they will be as follows −

A = 0011 1100 A&B = 0000 1100 A^B = 0011 0001

B = 0000 1101 A|B = 0011 1101 ~A = 1100 0011

Bitwise Operators

- Assume variable **A = 60** and variable **B = 13**, then –

A = 0011 1100
B = 0000 1101

Operator	Description	Example
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) = 12, i.e., 0000 1100
|	Binary OR Operator copies a bit if it exists in either operand.	(A | B) = 61, i.e., 0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) = 49, i.e., 0011 0001
~	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~A) = -60, i.e,. 1100 0100 in 2's complement form.
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 = 240 i.e., 1111 0000
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 = 15 i.e., 0000 1111

Assignment Operators

Operator	Description	Example
=	Simple assignment operator. Assigns values from right side operands to left side operand	C = A + B will assign the value of A + B to C
+=	Add AND assignment operator. It adds the right operand to the left operand and assign the result to the left operand.	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator. It subtracts the right operand from the left operand and assigns the result to the left operand.	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator. It multiplies the right operand with the left operand and assigns the result to the left operand.	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator. It divides the left operand with the right operand and assigns the result to the left operand.	C /= A is equivalent to C = C / A

Assignment Operators

Operator	Description	Example
%=	Modulus AND assignment operator. It takes modulus using two operands and assigns the result to the left operand.	C %= A is equivalent to C = C % A
<<=	Left shift AND assignment operator.	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator.	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator.	C &= 2 is same as C = C & 2
^=	Bitwise exclusive OR and assignment operator.	C ^= 2 is same as C = C ^ 2

Misc Operators

Operator	Description	Example
sizeof()	Returns the size of a variable.	sizeof(a), where a is integer, will return 4.
&	Returns the address of a variable.	&a; returns the actual address of the variable.
*	Pointer to a variable.	*a;
? :	Conditional Expression.	If Condition is true ? then value X : otherwise value Y

Operators Precedence

- Operator precedence determines the grouping of terms in an expression and decides how an expression is evaluated.
- Certain operators have higher precedence than others

```
x = 7 + 3 * 2
```

What is assigned to x?

Operators Precedence

- Operators with the highest precedence appear at the top of the table.
- Higher precedence operators will be evaluated first.

	Category	Operator	Associativity
1	Postfix	() [] -> . ++ - -	Left to right
2	Unary	+ - ! ~ ++ - - (type)* & sizeof	Right to left
3	Multiplicative	* / %	Left to right
4	Additive	+ -	Left to right
5	Shift	<< >>	Left to right
6	Relational	< <= > >=	Left to right
7	Equality	== !=	Left to right
8	Bitwise AND	&	Left to right

Operators Precedence

- Operators with the highest precedence appear at the top of the table.
- Higher precedence operators will be evaluated first.

	Category	Operator	Associativity
9	Bitwise XOR	^	Left to right
10	Bitwise OR	|	Left to right
11	Logical AND	&&	Left to right
12	Logical OR	||	Left to right
13	Conditional	?:	Right to left
14	Assignment	= += -= *= /= %= >>= <<= &= ^= |=	Right to left
15	Comma	,	Left to right

Expressions vs statements

`a + 7 + b` is an ***expression*** (has a value)

`a = b * c;` is a ***statement***

- ends in a semicolon
- also is an expression (value is value of `a`)

`a = b = c = 7;` is allowed

- Equivalent to `a = (b = (c = 7));`
- NOT `((a = b) = c) = 7;`

Comments

```
/* This is a lovely comment - comment on a single line. */


/*
 * Comments can span
 * multiple lines.
 */


// This is NOT a comment! - maybe… depends on the Compiler
```

- Comments provide clarity to the C source code allowing others to **better understand** what the code was intended to accomplish and greatly helping in **debugging** the code.

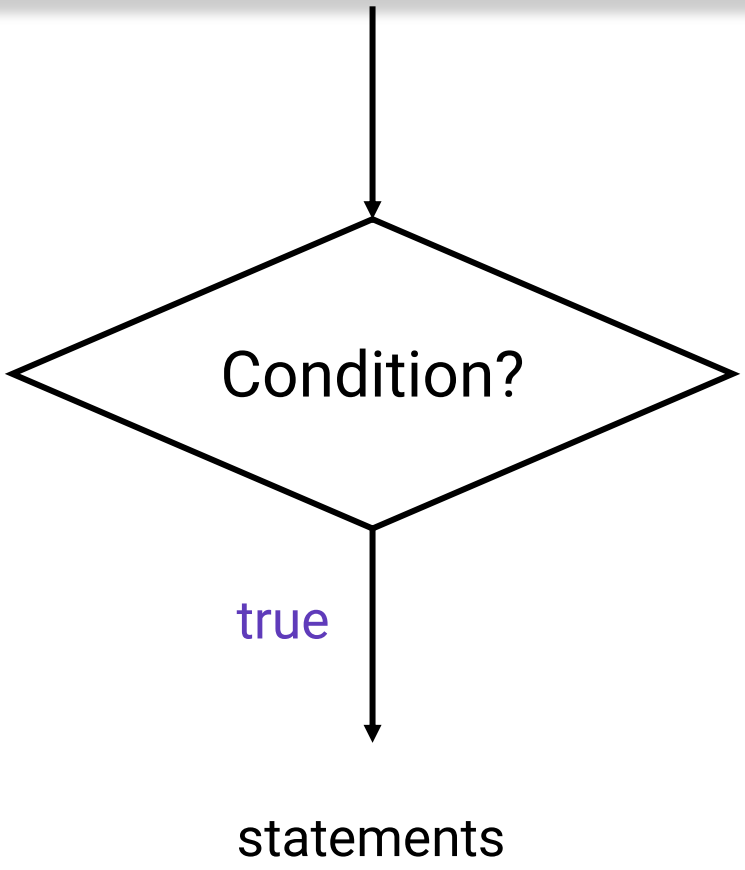
- Comments are especially important in **large projects** containing hundreds or thousands of lines of source code or in projects in which **many contributors** are working on the source code.
- It is important that you choose a **style** of commenting and use it **consistently** throughout your source code. Doing so makes the code more readable.

Decision Making

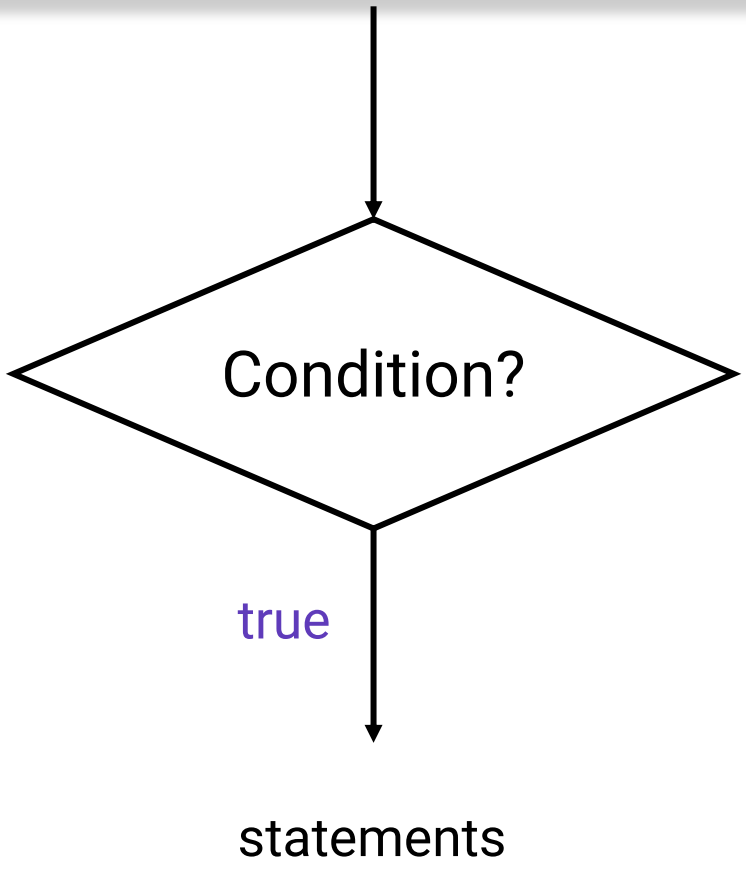
Decision Making

- `if`
- `if… else…`
- Nested if
- Condition Connectors
- Conditional Operator
- `switch` Statement
- Prefix and Postfix Operators

if

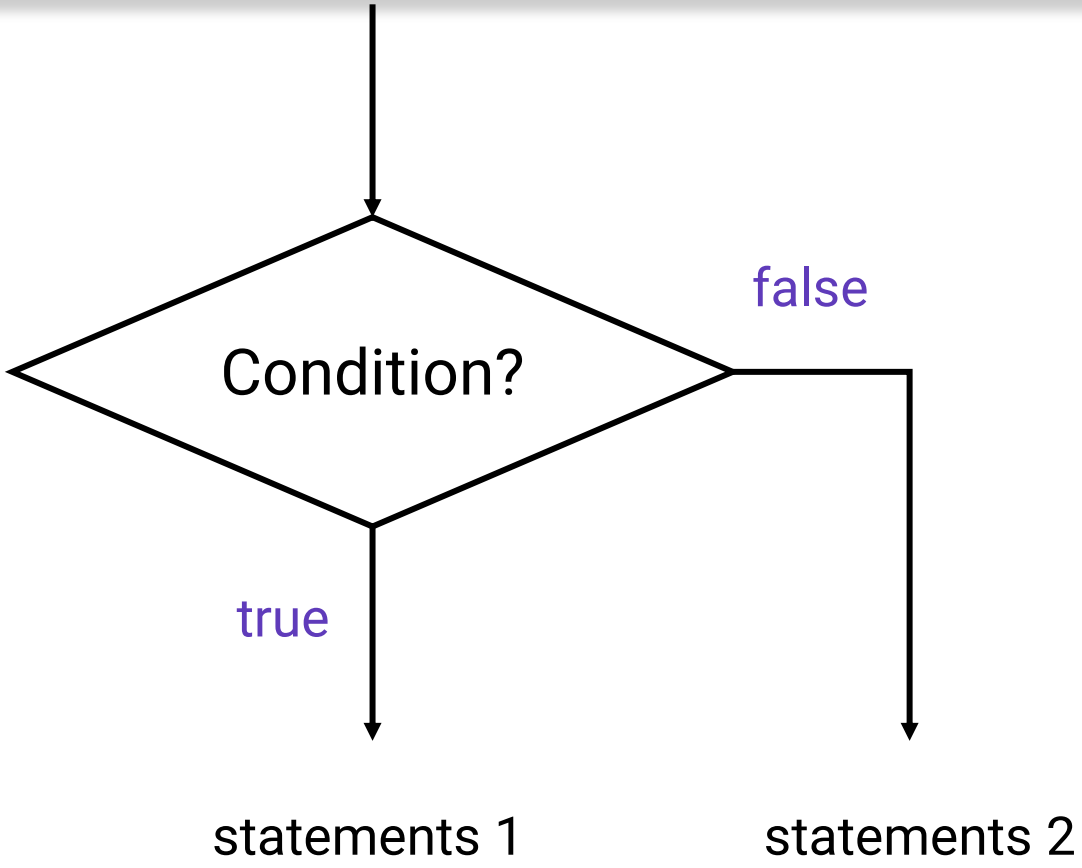


if




```
if (condition)
{
    …    // true_block
    …    // statements 1
}
```

if… else…




```
if (condition)
{
    …    // true_block
    …    // statements 1
}
else
{
    …    // false_block
    …    // statements 2
}
```

Example 1:

Write a program to print which variable is bigger in two variables.

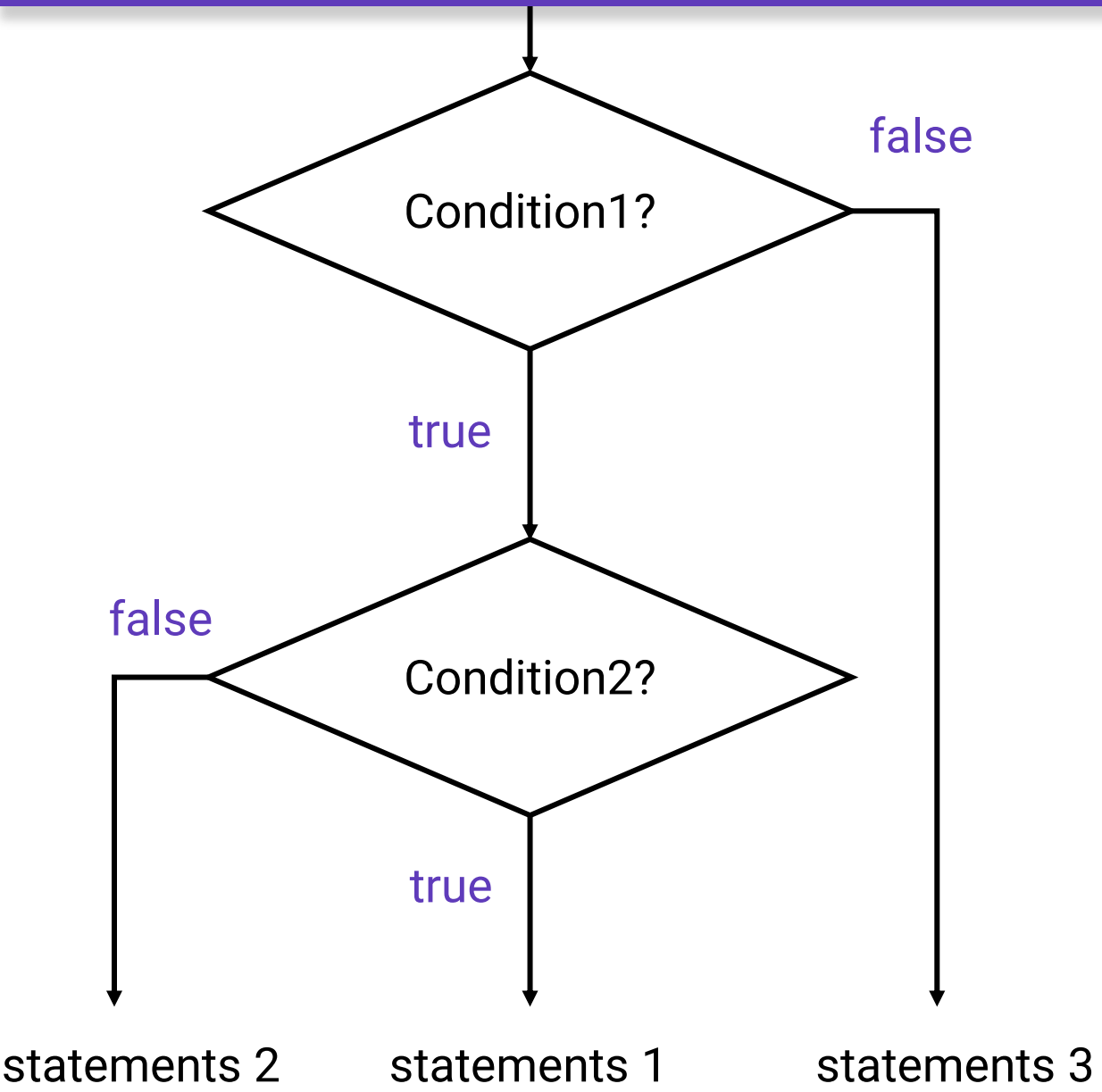
```
#include <stdio.h>
int main(void)
{
    int var1 = 12, var2 = 21;
    if( var1 > var2 )
    {
        printf("Variable 1 is bigger.\n");
    }
    else
    {
        printf("Variable 2 is bigger.\n");
    }
    return 0;
}
```

Example 2:

Write a program to check if a given number is even or odd.

```
#include <stdio.h>
int main(void)
{
    int var;
    printf("Enter a Number: ");
    scanf("%d", &var);
    if( var % 2 == 0 )
    {
       printf("You entered an Even Number.\n");
    }
    else
    {
       printf("You entered an Odd Number\n");
    }
    return 0;
}
```

Nested if




```
if( condition1 )
{
    if( condition2 )
    {
        …    // statements 1
    }
    else
    {
        …    // statements 2
    }
}
else
{
    …        // statements 3
}
```

Example 3:

Write a program to find Maximum out of three numbers.

```c
#include <stdio.h>
int main(void)
{
    int a, b, c;
    printf("Enter three integers: ");
    scanf("%d%d%d", &a, &b, &c);
    if( a > b )
    {
        if( a > c )
        {
            printf("%i is the largest.\n", a);
        }
        else
        {
            printf("%i is the largest.\n", c);
        }
    }
    else
    {
        if( b > c )
        {
            printf("%i is the largest.\n", b);
        }
        else
        {
            printf("%i is the largest.\n", c);
        }
    }
    return 0;
}
```

Note: C assumes any non-zero and non-null values as true, and if it is either zero or null, then it is assumed as false value.

```
if ( 'a' )
{
    printf(‘yup');
}
else
{
    printf('nope');
}
```

Condition Connectors

Operator Example	Treatment
`&&`	`if(a>5 && a<10)` executes True block if both the conditions are satisfied, otherwise False_block is executed.
`||`	`if(a>5 || a<10)` executes True_block if any of the condition is satisfied.

Example 4:

Difference between `&&` and `||` in `if`

```
#include <stdio.h>
int main(void)
{
    int a = 10, b = 20;
    if( a > 5 && b > 10 )
    {
        printf("Both conditions are
                satisfied.\n");
    }
    if( a > 15 || b > 15 )
    {
        printf("Anyone condition is
                compulsory true.\n");
    }
    return 0;
}
```

Conditional Operator

```
var = (conditional_statement) ? True_block/expression :False_block/expression;
```

Example 5:

Write a program to find Maximum out of three numbers.

```
#include <stdio.h>
int main(void)
{
    int a, b, c, max;
    printf("Enter three numbers: ");
    scanf("%d%d%d", &a, &b, &c);
    max = ( a > b ) ? a : b;
    max = ( max > c ) ? max : c;
    printf("Maximum value = %d\n", max);
    return 0;
}
```

Example 6:

Write a program to give bonus according to grade of employees.

Employee Grade	Bonus
1	£2,000
2	£3, 000
3	£5, 000

```
#include <stdio.h>
int main(void)
{
    int salary, grade, basis;
    printf("Enter basic salary and grade: ");
    scanf("%d%d", &basis, &grade);
    salary = basis +
    ((grade==1)?200:((grade==2)?300:500));
    printf("Total salary is %d.\n", salary);
    return 0;
}
```

What if conditions increased?

switch Statement

Switch is a method which is going to take an integer parameter and depending upon that parameter will do the given operations.

```
switch (integer_expression)
{
    case value:
        statements;
        break;
    case value:
        statements;
        break;
    default:
        statements;
        break;
}
```

Example 7:

Write a program to determine grades for a student in two tests.

Avg. Score	Grade
>= 80	Distinction
>= 60	First Division
>= 50	Second Division
>= 40	Pass
otherwise	Fail

```c
#include <stdio.h>
int main(void)
{
    int score1, score2;
    printf("Enter scores for two tests: ");
    scanf("%d%d", &score1, &score2);
    switch( (score1 + score2) / 2 / 10 )
    {
        case 10:
        case 9:
        case 8:
            printf("Distinction.\n");
            break;
        case 7:
        case 6:
            printf("First Division.\n");
            break;
        case 5:
            printf("Second Division.\n");
            break;
        case 4:
            printf("Pass.\n");
            break;
        default:
            printf("Fail.\n");
    }
    return 0;
}
```

Prefix and Postfix Operators

Prefix: If there are more than one operation to be carried in an expression and anyone of them is a **prefix operation** then that will be **the first operation** to be carried out.

Postfix: If there are more than one operation to be carried in an expression and any one of them is a **postfix operation** then that will be **the last operation** to be carried out.

Example 8.1:

```
#include <stdio.h>
int main(void)
{
    int x = 56, y = 65;
    if ( --x > 50 && ++y > 50)
    {
        printf("x=%d, y=%d\n", x, y );
    }
    return 0;
}
```

Result:

A. x=55, y=65

B. x=55, y=66

C. x=56, y=65

D. x=56, y=66

Example 8.2:

```
#include <stdio.h>
int main(void)
{
    int x = 56, y = 65;
    if ( --x > 50 || ++y > 50)
    {
        printf("x=%d, y=%d\n", x, y );
    }
    return 0;
}
```

Result:

A. x=55, y=65

B. x=55, y=66

C. x=56, y=65

D. x=56, y=66

Example 8.1 & 8.2:

```c
#include <stdio.h>
int main(void)
{
    int x = 56, y = 65;
    if ( --x > 50 && ++y > 50)
    {
        printf("x=%d, y=%d\n", x, y );
    }
    return 0;
}
```

Result: x=55, y=66

```c
#include <stdio.h>
int main(void)
{
    int x = 56, y = 65;
    if ( --x > 50 || ++y > 50)
    {
        printf("x=%d, y=%d\n", x, y );
    }
    return 0;
}
```

Result: x=55, y=65

(Because in || (OR) if the first condition is satisfied then the compiler is not going to check the second condition and hence the **y** is not incremented and remains 65.)

Summary

Today

- Basic I/O
  - `printf( )`, `scanf( )`
- Operators
  - Arithmetic, relational, logical, bitwise, assignment, misc
- Decision Making
  - `if`, `if… else…` , nested `if`, condition connectors, conditional operator, `switch` statement, prefix and postfix operators

Next

- C Language Basics
  - Loops
- Online Judge
  - https://student.csc.liv.ac.uk/JudgeOnline