
Week 2

XMUT-NWEN 241 - 2026 T2

Systems Programming

Agatha Rachmat

School of Engineering and Computer Science

Victoria University of Wellington

Basic I/O

Content

- Basic I/O
- Control flow
 - If-else
 - Else-if
 - Switch
- Iterations
 - While-loop
 - For-loop
 - Do-while-loop
- Same syntax as Java

I/O Using Standard C Library

- Recall: C provides a set of header files (standard C library) that you can use to write your code

C provides a standard library which consists of the following headers:

<code>assert.h</code>	<code>float.h</code>	<code>math.h</code>	<code>stdarg.h</code>	<code>stdlib.h</code>
<code>ctype.h</code>	<code>limits.h</code>	<code>setjmp.h</code>	<code>stddef.h</code>	<code>string.h</code>
<code>errno.h</code>	<code>locale.h</code>	<code>signal.h</code>	<code>stdio.h</code>	<code>time.h</code>

- You don't have to start from scratch!

I/O Streams

- C provides functions with input and output capability
- From the program's point of view, data input and data output are made possible through files
- Every C program has access to 3 such files: `stdin`, `stdout`, `stderr`

File	Description	Remarks
<code>stdin</code>	Standard input file	Connected to the keyboard
<code>stdout</code>	Standard output file	Connected to the screen
<code>stderr</code>	Standard error file	Connected to the screen

I/O Functions

- C input/output functions can be classified into 2 types:
 - Non-formatted input/output
 - getchar
 - putchar
 - gets
 - puts
 - Formatted input/output
 - `printf` and its variants
 - `scanf` and its variants

How To Use a Function

- Find its manual or documentation
 - In Linux terminal, use the man command
 - You can also search online
 - This website provides a pretty good documentation for the standard C library:
https://www.tutorialspoint.com/c_standard_library/index.htm
- What to look for in the function manual?
 - What the function does
 - What header file(s) to include
 - What are the arguments to the function
 - What is the return type
 - What happens in case of errors

printf()

- `printf()` writes a string to the standard output stream (`stdout`)
 - The string is formatted using additional arguments that follow the initial string.
 - `%d` format specifier to display the value of an integer variable.
 - `%c` to display character,
 - `%f` to display float variable,
 - `%s` to display string variable
 - To generate a newline, we use “`\n`” in C `printf()` statement.

```
char ch = 'A';  
printf("Character is %c \n", ch);
```

Format specifiers in C

Format Specifier	Type
%c	Character
%d	Signed integer
%u	Unsigned int
%e or %E	Scientific notation of floats
%f	Float values
%hi	Signed integer (short)
%ld	Long
%lf	Double
%Lf	Long double
%lli or %lld	Long long
%o	Octal representation
%p	Pointer
%s	String
%x or %X	Hexadecimal representation
%%	Prints % character

scanf()

- scanf() accepts input from the standard input stream (stdin).
 - The format of the expected items is specified, and it returns the number of items successfully scanned
 - The *format specifier* %d is used in scanf() statement. So, the value entered is received as an integer and %s for string.
 - Ampersand (&) is used before the variable name in scanf() statement.

```
char ch;  
scanf("%c", &ch);
```

Control flow

Control flow: if-else statement

```
if (expression){  
    statement  
}
```

- If expression evaluates to true, statement is executed

*Recall: true $\rightarrow$ non-zero;
false $\rightarrow$ zero

```
if (x != 0.0)  
    y /= x;  
  
if (c == ' ') {  
    ++blank_counter;  
    printf("Found another blank\n");  
}
```

```
if (a > b)  
    max = a;  
else  
    max = b;
```

Conditional expression (ternary operator)

$$expr_1 \text{ ? } expr_2 \text{ : } expr_3$$

- $expr_1$ is evaluated first
- If $expr_1$ evaluates to true, then expression $expr_2$ is evaluated and that is used as the value of the expression
- Otherwise, $expr_3$ is evaluated and that is used as the value of the expression
- Example:

```
z = (a > b) ? a : b;  /* z = max(a, b) */
```

Boolean expressions

What can go in the condition of an **if** statement?

- Boolean expressions:
 - numeric comparisons: $(x > 0)$ $(\text{day} \leq 7)$,
 $(x == y)$, $(\text{day} \neq 7)$
 - logical operators: `!`, `&&`, `||` (not, and, or)
 $(x > 0 \ \&\& \ x < 7)$

Writing Boolean expressions

Mostly, boolean expressions are straightforward,
There are just a few things to remember:

- `==` is the "equals" operator for simple values,
`=` is assignment

`(age == 15)`

vs

~~`(age = 15 );`~~

- But only use `==` for numbers (or characters, or references)

Using else-if statement

- Can put another **if** statement in the **else** part:

```
if ( <condition1> ) {  
    <actions to perform if condition1 is true>  
    :  
}  
else if ( <condition2> ) {  
    <actions to perform if condition 2 is true (but not condition 1)>  
    :  
}  
else if ( <condition3> ) {  
    <actions to perform if condition 3 is true (but not conditions 1, 2)>  
    :  
}  
else {  
    <actions to perform if other conditions are false>  
}
```

Traps with Boolean expressions

- When combining with `&&` and `||`, which binds tighter?

if (`x > 5 && y <= z || day == 0`) {

- Use (and) whenever you are not sure!

if ((`x > 5 && y <= z`) `|| day == 0`) { ...

if (`x > 5 && ( y <= z || day == 0 )`) { ...

- The not operator `!` goes in front of expressions:

• **if** (`!(x > 5 && y <= z)`) { ...

NOT if (`(x !> 5 && y !<= z)`)

<code>!</code>	→ highest
<code>&&</code>	→ next
<code> </code>	→ lower

Example: else-if statement

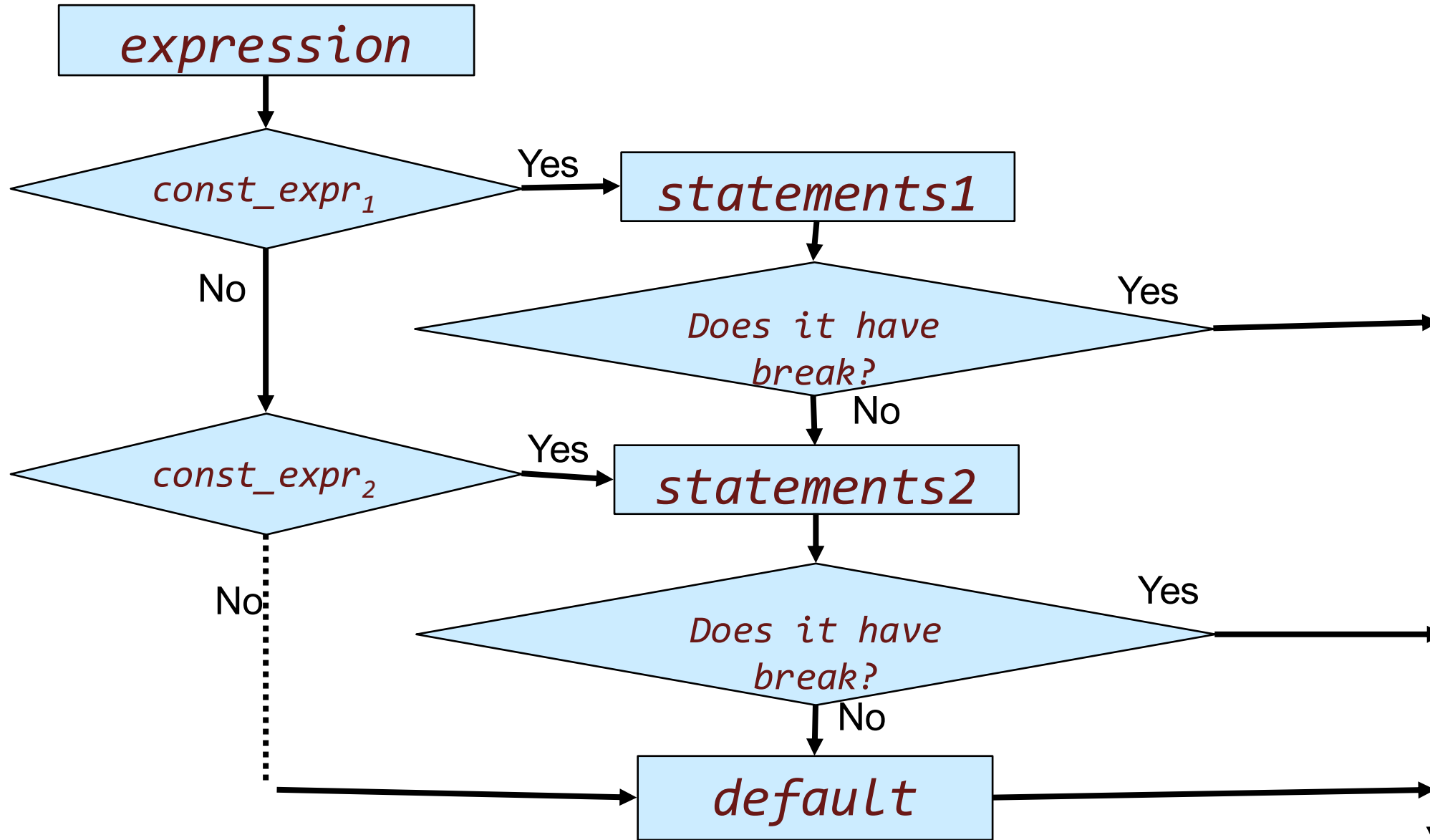
```
if (temp <= 0) {  
    printf("It's freezing out there.\n");  
} else if (temp <= 10) {  
    too_cold++;  
    printf("It's too cold for me.\n");  
} else if (temp <= 20)  
    printf("It's still cold.\n");  
} else {  
  
    printf("Awesome!\n");  
}
```

Control flow: switch statement

```
switch (expression) {  
    case const_expr1:  
        statements1  
        break;  
    case const_expr2:  
        statements2  
    ...  
    case const_exprN:  
        statementsN  
    default:  
        statements  
}
```

- The default part is optional
- **const_expr**₁ to **const_expr**_N must be integer constants or constant expressions
- If *expression* matches **const_expr**_k, execution starts at that case
- **default** is executed if none of the cases match
- The statements can consist of single or multiple statements, or compound statements

Switch statement



Example: switch statement

```
char c = getchar();

switch(c) {
    case 'Y':
    case 'y':
        printf("You answered yes.\n");
        break;
    case 'N':
    case 'n':
        printf("You answered no.\n");
        break;
    default:
        printf("What was that?\n");
        break;
}
```

Is this necessary?

Iterations

Iteration: while-loop statement

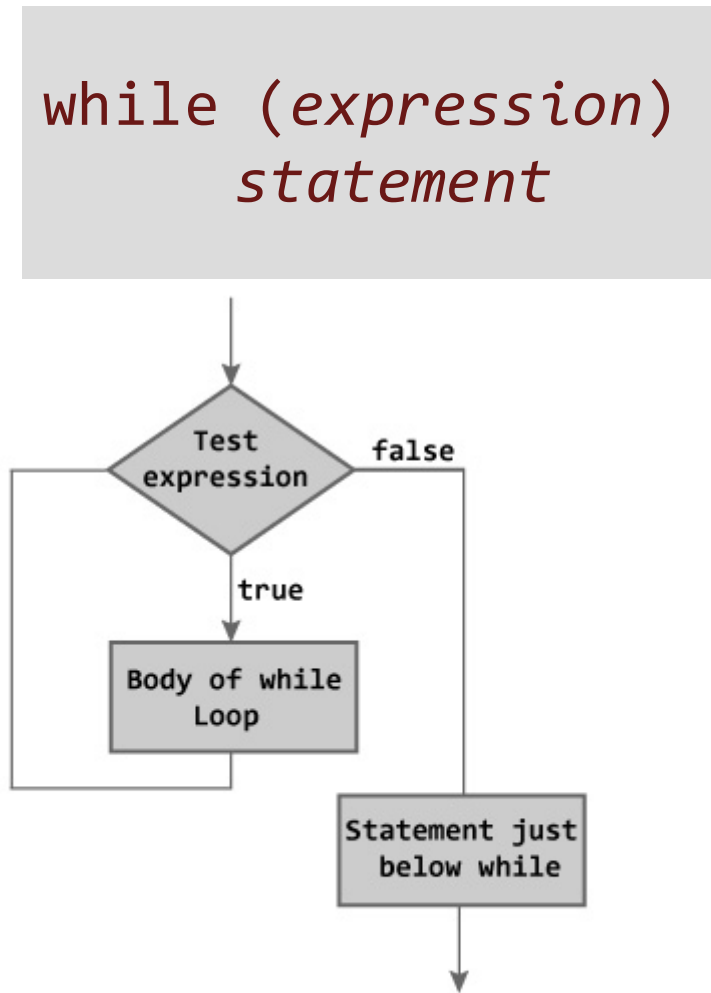


Figure: Flowchart of while Loop

- If `expression` evaluates to true:
 - `statement` is executed
 - `expression` is re-evaluated again
- Cycle continues until `expression` evaluates to false
- `statement` can be single or compound statement

Iteration: for-loop statement

```
for (expr1; expr2; expr3)  
  statement
```



```
expr1;  
while (expr2) {  
  statement  
  expr3;  
}
```

- The expressions are optional
- *expr*₁ and *expr*₃ are usually assignments or function calls
- *expr*₂ is usually a relational expression
 - If *expr*₂ is missing, it is taken as permanently true

Iteration: do-while-loop statement

```
do {  
    statement  
}  
while (expression);
```

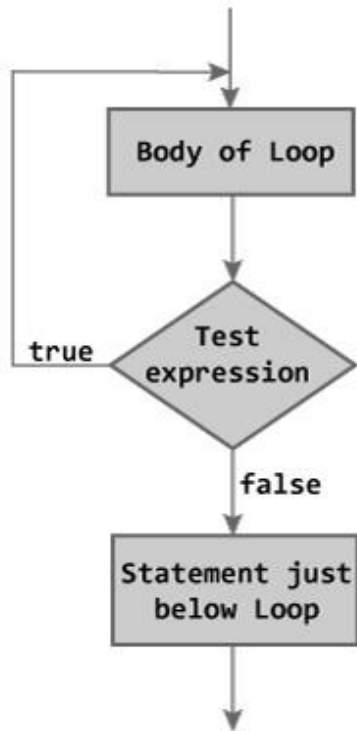


Figure: Flowchart of do...while Loop

- statement is executed, then expression is evaluated
- If expression evaluates to true, statement is executed again
- Loop terminates when expression evaluates to false

Example: loop statements

Infinite loops:

```
while(1);
```

```
for (;;);
```

```
do {} while(1);
```

```
int i = 10;
while(i > 0) {
    printf("%d\n", i);
    i--;
}
```

```
for(int i = 10; i > 0; i--) {
    printf("%d\n", i);
}
```

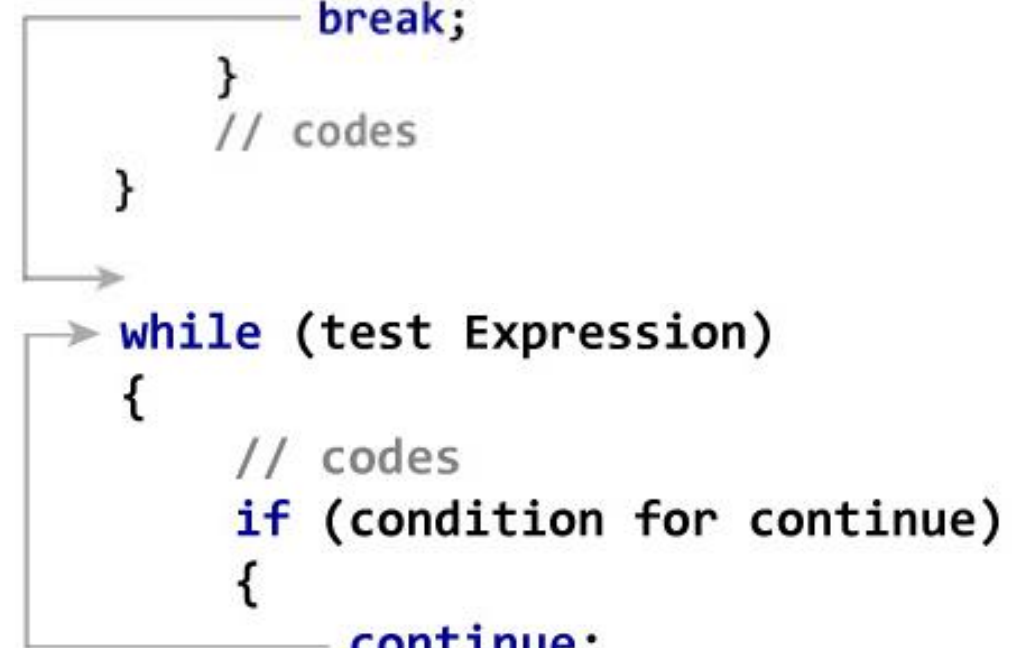
```
do {
    printf("Do you agree with the contract?\n");
    ans = getchar();
} while (ans != 'Y' || ans != 'y');
```

Statements that can alter control flow & loop

- **break, return and continue**
 - **break**: jumps out of the loop or switch
 - **return**: jumps out of the loop or switch (the loop or switch must be inside a function)
 - **continue**: stops current loop iteration and starts next iteration

```
while (test Expression)
{
    // codes
    if (condition for break)
    {
        break;
    }
    // codes
}

while (test Expression)
{
    // codes
    if (condition for continue)
    {
        continue;
    }
    // codes
}
```



Differences

Condition in if-else, else-if, while-loop, for-loop and do-while-loop

- In Java, the condition must be an expression that evaluates to boolean
- In C, the condition is an expression that evaluates to any type
 - Considered true if expression evaluates to non-zero value, otherwise false

Break and continue

- In Java, **break** and **continue** statements can be labelled or unlabelled
- In C, **break** and **continue** statements do not support labels

Example

```
int i = 100;

while (i--) {
    // do stuff
}
```

- Valid in C
- Will generate syntax error in Java
 - Condition inside while-loop should be changed to an expression that will evaluate to boolean type, e.g. `i-- > 0`

Example

Java

Java requires the condition to be a boolean expression:

```
int x = 5;

if (x > 0) {

    System.out.println("
    Positive");
}
```

$x > 0$ produces either: True or false
You **cannot** do this in Java
because x is an int, not a boolean.

C

In C, the condition can be an integer:

```
int x = 5;

if (x) {
    printf("True\n");
}
```

and 5 is **non-zero**, C considers it **true**.

Next Lecture

- Function