



PROGRAMMING METHODOLOGY (PHƯƠNG PHÁP LẬP TRÌNH)

UNIT 6: Repetition Statements

Acknowledgement

- The contents of these slides have origin from School of Computing, National University of Singapore.
- We greatly appreciate support from Mr. Aaron Tan Tuck Choy for kindly sharing these materials.

Policies for students

- These contents are only used for students PERSONALLY.
- Students are NOT allowed to modify or deliver these contents to anywhere or anyone for any purpose.

Recording of modifications

- Currently, there are no modification on these contents.

Unit 6: Repetition Statements

Objectives:

- Using repetition structure to repeat some action until a terminating condition is reached
- Using different types of repetition structure

Reference:

- Chapter 4 Lessons 4.7 – 4.11

Unit 6: Repetition Statements (1/2)

1. Loops!
2. The *while* loop
 - 2.1 Demo
 - 2.2 Loop Condition
 - 2.3 Style: Indentation
3. The *do-while* loop
4. The *for* loop
5. Example: Odd Integers
6. Common Errors
7. Some Notes of Caution

Unit 6: Repetition Statements (2/2)

- 8. Using *break* in Loop
- 9. Using *continue* in Loop

Recall: Control Structures

Sequence 

Selection  if-else, switch

Repetition

1. LOOPS! (1/2)



“A program without a loop and a structure variable isn’t worth writing.”

Alan J. Perlis

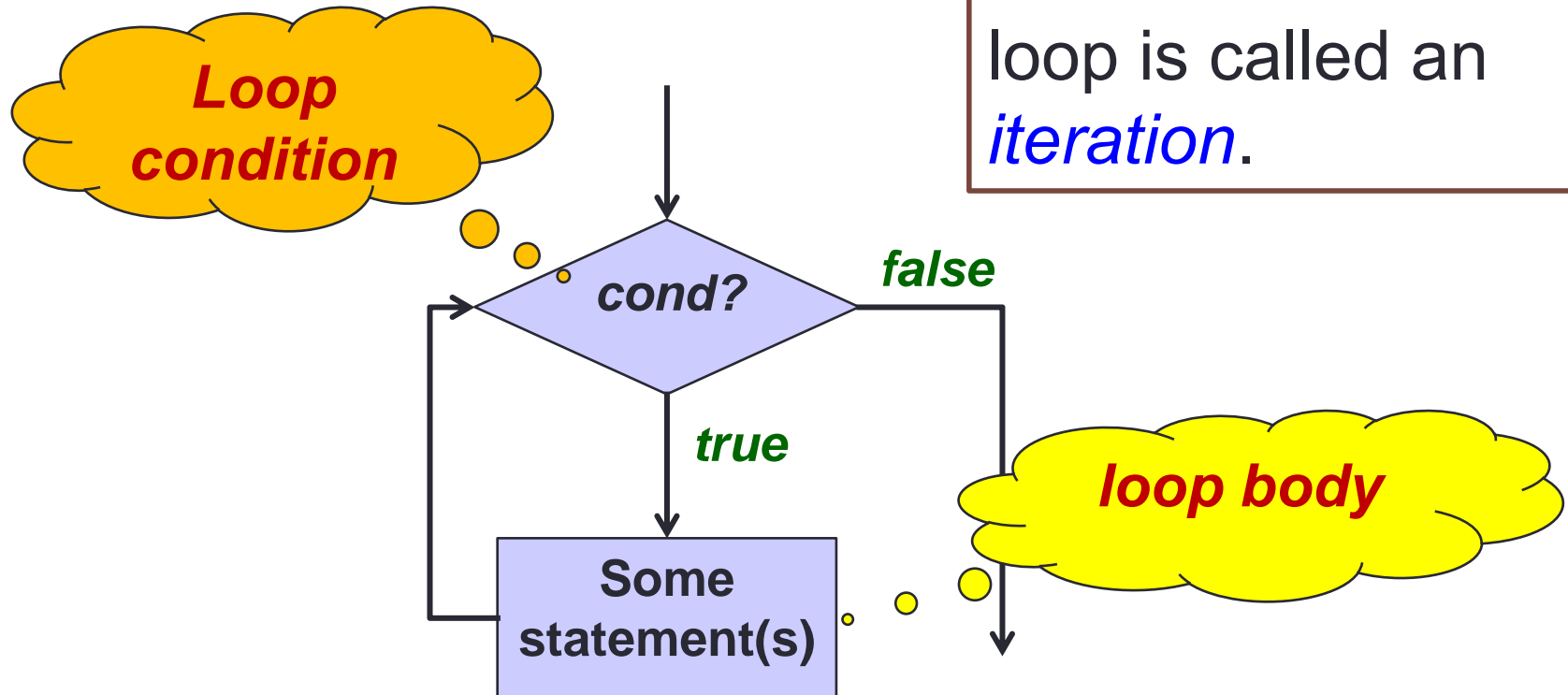
Yale University

The first recipient of ACM Turing Award

- A **loop** is a statement whose job is to **repeatedly** execute some other statement(s).

1. LOOPS! (2/2)

Each round of the loop is called an *iteration*.



1. Loop: Demo (1/3)

- Keep prompting the user to input a non-negative integer, and output that integer.
- Halt the loop when the input is negative.
- **Key observations:**
 - You keep repeating a task while certain condition is met, or alternatively, you repeat until the condition is not met.
 - You do not know beforehand how many iterations there will be.

Enter a number: 12

You entered: 12

Enter a number: 0

You entered: 0

Enter a number: 26

You entered: 26

Enter a number: 5

You entered: 5

Enter a number: -1

1. Loop: Demo (2/3)

Algorithm:

read num

condition

if (num >= 0) {

print the value entered
read num

body

}
else end input request

if (num >= 0) {

print the value entered
read num

}

else end input request

....

Same code repeated

Enter a number: 12

You entered: 12

Enter a number: 0

You entered: 0

Enter a number: 26

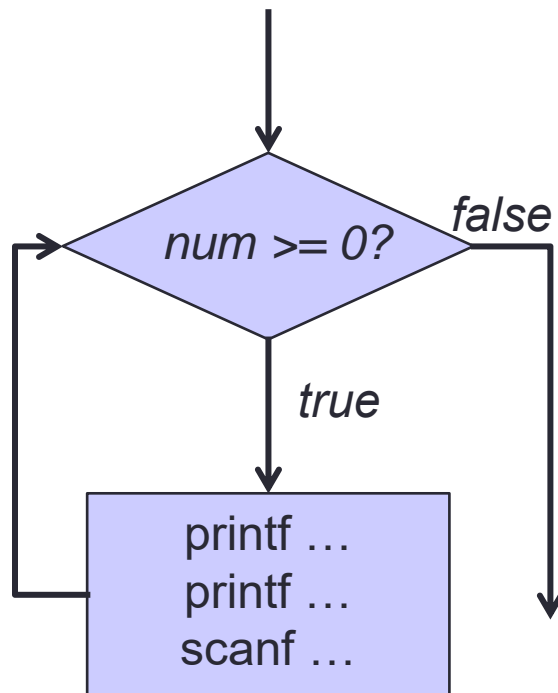
You entered: 26

Enter a number: 5

You entered: 5

Enter a number: -1

1. Loop: Demo (3/3)



Unit6_ReadandPrint.c

```
#include <stdio.h>

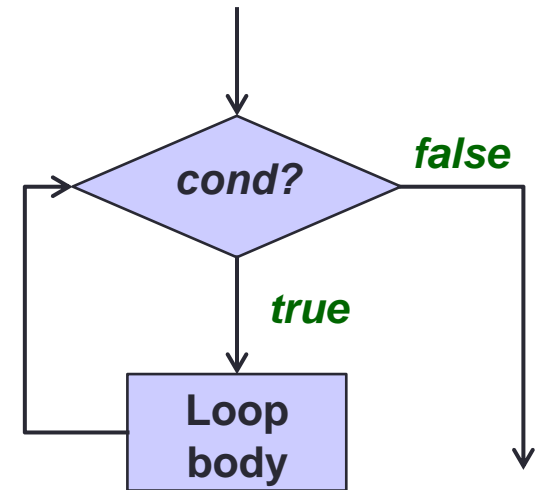
int main(void) {
    int num;

    printf("Enter a number: ");
    scanf("%d", &num);
    while (num >= 0) {
        printf("You entered: %d\n", num);
        printf("Enter a number: ");
        scanf("%d", &num);
    }

    return 0;
}
```

2. The *while* Loop

```
while ( condition )  
{  
    // loop body  
}
```



If condition is **true**, execute loop body; otherwise, terminate loop.

2.1 The *while* Loop: Demo (1/3)

- Keep prompting the user to input a non-negative integer, and print that integer.
- Halt the loop when the input is negative.
- Print the maximum integer input.

Enter a number: 12

Enter a number: 0

Enter a number: 26

Enter a number: 5

Enter a number: -1

The maximum number is 26

2.1 The *while* Loop: Demo (2/3)

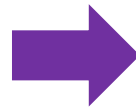
```
maxi = 0;  
read num;
```

```
if (num >= 0) {  
    if (maxi < num)  
        maxi = num;  
    read num;  
}  
else stop;
```

```
if (num >= 0) {  
    if (maxi < num)  
        maxi = num;  
    read num;  
}  
else stop;
```

```
...
```

```
print maxi;
```



```
maxi = 0;  
read num;  
while (num >= 0) {  
    if (maxi < num)  
        maxi = num;  
    read num;  
}  
  
print maxi;
```


2.1 The *while* Loop: Demo (3/3)

Unit6_FindMax.c

```
#include <stdio.h>

int main(void) {
    int num, maxi = 0;

    printf("Enter a number: ");
    scanf("%d", &num);
    while (num >= 0) {
        if (maxi < num) {
            maxi = num;
        }
        printf("Enter a number: ");
        scanf("%d", &num);
    }
    printf("The maximum number is %d\n", maxi);

    return 0;
}
```

2.2 Condition for *while* Loop: (1/2)

```
// pseudo-code  
a = 2;  
b = 7;  
while (a == b) {  
    print a;  
    a = a + 2;  
}
```

Output: ?

- When the loop condition is always **false**, the loop body is not executed.

2.2 Condition for *while* Loop: (2/2)

```
// pseudo-code  
a = 2;  
b = 7;  
while (a != b) {  
    print a;  
    a = a + 2;  
}
```

Output: ?

2
4
6
8
10
:
:

Ctrl-c to
interrupt

- When the loop condition is always **true**, the loop body is executed forever – **infinite loop**.

2.3 Style: Indentation for *while* Loop

- Loop body must be indented.
- Comment in loop body must be aligned with statements in loop body.
- Closing brace must be on a line by itself and aligned with the *while* keyword.

```
while (cond) {  
    // loop body  
    statement-1;  
    statement-2;  
    ...  
}
```

or

```
while (cond)  
{  
    // loop body  
    statement-1;  
    statement-2;  
    ...  
}
```

```
while (cond) {  
// loop body  
statement-1;  
...  
}
```

No indentation!



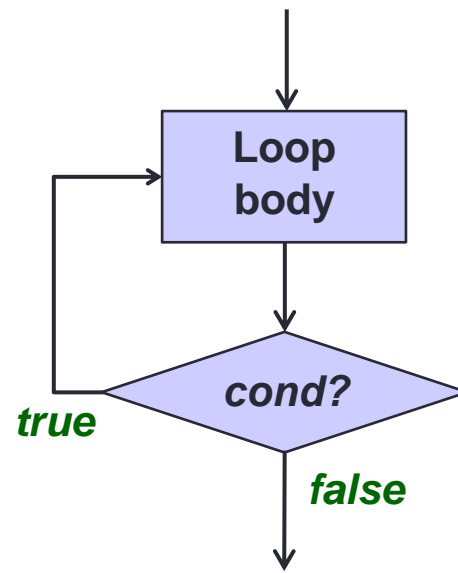
```
while (cond) {  
    // loop body  
    statement-1;  
    statement-2; }
```



3. The *do-while* Loop (1/3)

```
do  
{  
    // loop body  
} while ( condition );
```

Execute loop body
at least once.



3. The *do-while* Loop (2/3)

- Example: Count the number of digits in an integer.

```
do
{
    // loop body
} while ( condition );
```

Unit6_CountDigits.c

```
// Precond: n > 0
int count_digits(int n) {
    int counter = 0;


    do {
        counter++;
        n /= 10;
    } while (n > 0);

    return counter;
}
```

3. The *do-while* Loop (3/3)


- Style: similar to *while* loop

```
do {  
    // loop body  
    statement-1;  
    statement-2;  
} while (cond);
```



or

```
do  
{  
    // loop body  
    statement-1;  
    statement-2;  
} while (cond);
```



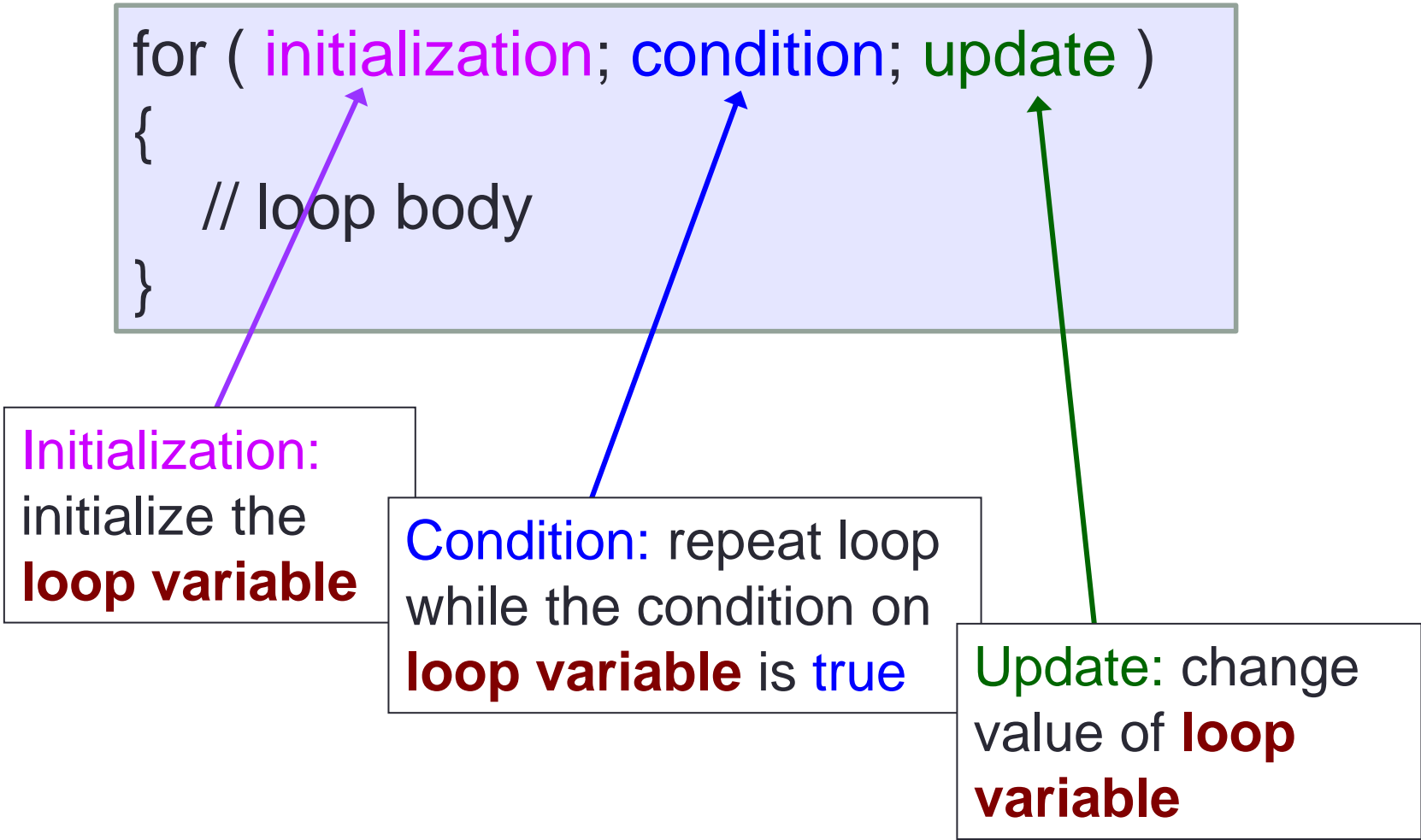
```
do {  
    // loop body  
    statement-1;  
    statement-2;  
} while (cond);
```

No indentation!



4. The *for* Loop (1/2)

```
for ( initialization; condition; update )  
{  
    // loop body  
}
```



Initialization:
initialize the
loop variable

Condition: repeat loop
while the condition on
loop variable is **true**

Update: change
value of **loop
variable**

4. The *for* Loop (2/2)

- Example: Print numbers 1 to 10

```
int n;  
for (n=1; n<=10; n++) {  
    printf("%3d", n);  
}
```

Steps:

1. `n=1;`

2. `if (n<=10) {`
 `printf(...);`
 `n++;`
 Go to step 2
}

3. Exit the loop

5. Example: Odd Integers (1/2)

Unit6_OddIntegers_v1.c

```
#include <stdio.h>
void print_odd_integers(int) ;
int main(void) {
    int num;
    printf("Enter a positive integer: ");
    scanf("%d", &num);
    print_odd_integers(num);
    return 0;
}

// Precond: n > 0
void print_odd_integers(int n) {
    int i;
    for (i=1; i<=n; i+=2)
        printf("%d ", i);
    printf("\n");
}
```

5. Example: Odd Integers (2/2)

Unit6_OddIntegers_v2.c

```
// Precond: n > 0
void print_odd_integers(int n) {
    int i;
    for (i=1; i<=n; i++)
        if (i%2 != 0)
            printf("%d ", i);
    printf("\n");
}
```

Unit6_OddIntegers_v3.c

```
// Precond: n > 0
void print_odd_integers(int n) {
    for ( ; n > 0; n--)
        if (n%2 != 0)
            printf("%d ", n);
    printf("\n");
}
```

Empty
statement

*Values printed from
largest to smallest.*



6. Common Errors (1/2)

- What are the outputs for the following programs? (Do not code and run them. Trace the programs manually.)
- We will discuss this in class.

```
int i;  
  
for (i=0; i<10; i++);  
    printf("%d\n", i);
```

Unit6_CommonErrors1.c

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

Unit6_CommonErrors2.c



6. Common Errors (2/2)

```
int z = 3;  
while (z = 1) {  
    printf("z = %d\n", z);  
    z = 99;  
}
```

Unit6_CommonErrors3.c

- **Off-by-one error**; make sure the loop repeats exactly the correct number of iterations.
- Make sure the loop body contains a statement that will eventually cause the loop to terminate.
- Using '=' where it should be '=='
- Putting ';' where it should not be (just like for the 'if' statement)



7. Some Notes of Caution (1/2)

- Involving real numbers
 - Trace the program manually without running it.

```
double one_seventh = 1.0/7.0;  
double f = 0.0;  
  
while (f != 1.0) {  
    printf("%f\n", f);  
    f += one_seventh;  
}
```

Unit6_Caution1.c



7. Some Notes of Caution (2/2)

- Involving 'wrap-around';
 - Trace the program manually without running it.

```
int a = 2147483646;  
int i;  
  
for (i=1; i<=5; i++) {  
    printf("%d\n", a);  
    a++;  
}
```

Unit6_Caution2.c

8. Using *break* in Loop (1/4)

- You have seen *break* in switch statement
- *break* can also be used in a loop
- Test out [Unit6_BreakInLoop.c](#)

8. Using *break* in Loop (2/4)

```
// without 'break'
printf ("Without 'break':\n");
for (i=1; i<=5; i++) {
    printf("%d\n", i);
    printf("Ya\n");
}
```

Without 'break':

```
1
Ya
2
Ya
3
Ya
4
Ya
5
Ya
```

```
// with 'break'
printf ("With 'break':\n");
for (i=1; i<=5; i++) {
    printf("%d\n", i);
    if (i==3)
        break;
    printf("Ya\n");
}
```

With 'break':

```
1
Ya
2
Ya
3
```

8. Using *break* in Loop (3/4)

```
// with 'break' in a nested loop
printf("With 'break' in a nested loop:\n");
for (i=1; i<=3; i++) {
    for (j=1; j<=5; j++) {
        printf("%d, %d\n", i, j);
        if (j==3)
            break;
        printf("Ya\n");
    }
}
```

- In a nested loop, *break* only breaks out of the inner-most loop that contains it.

With ...

1, 1

Ya

1, 2

Ya

1, 3

2, 1

Ya

2, 2

Ya

2, 3

3, 1

Ya

3, 2

Ya

3, 3

8. Using *break* in Loop (4/4)

- Use *break* sparingly, because it violates the one-entry-one-exit control flow.
- A loop with *break* can be rewritten into one without *break*.

```
// with break
int n, i = 1, sum = 0;

while (i <= 5) {
    scanf("%d", &n);
    if (n < 0)
        break;
    sum += n;
    i++;
}
```

```
// without break
int n, i = 1, sum = 0;
int isValid = 1;
while ((i <= 5) && isValid) {
    scanf("%d", &n);
    if (n < 0)
        isValid = 0;
    else {
        sum += n;
        i++;
    }
}
```

9. Using *continue* in Loop (1/3)

- Test out [Unit6_ContinueInLoop.c](#)
- *continue* is used even less often than *break*

9. Using *continue* in Loop (2/3)

```
// without 'continue'
printf ("Without 'continue':\n");
for (i=1; i<=5; i++) {
    printf("%d\n", i);
    printf("Ya\n");
}
```

Without 'continue':

```
1
Ya
2
Ya
3
Ya
4
Ya
5
Ya
```

```
// with 'continue'
printf ("With 'continue':\n");
for (i=1; i<=5; i++) {
    printf("%d\n", i);
    if (i==3)
        continue;
    printf("Ya\n");
}
```

The rest of the loop body is skipped if (i==3), and continue to next iteration.

With 'continue':

```
1
Ya
2
Ya
3
4
Ya
5
Ya
```

9. Using *continue* in Loop (3/3)

```
// with 'continue' in a nested loop
printf("With 'continue' in a nested loop:\n");
for (i=1; i<=3; i++) {
    for (j=1; j<=5; j++) {
        printf("%d, %d\n", i, j);
        if (j==3)
            continue;
        printf("Ya\n");
    }
}
```

- In a nested loop, *continue* only skips to the next iteration of the inner-most loop that contains it.

```
3, 1
Ya
3, 2
Ya
3, 3
3, 4
Ya
3, 5
Ya
```

With ...

```
1, 1
Ya
1, 2
Ya
1, 3
1, 4
Ya
1, 5
Ya
2, 1
Ya
2, 2
Ya
2, 3
2, 4
Ya
2, 5
Ya
```

Summary

- In this unit, you have learned about
 - The use of *if-else* construct and *switch* construct to alter program flow
 - The use of relational and logical operators
 - Style issues such as indentation, naming of boolean flags and replacing *if* statement with an assignment statement
 - The use of *break* and *continue* in a loop
 - How to test a selection construct with exhaustive test data, and to ensure that all alternative paths in the selection construct are examined

End of File