EEE101: C Programming & Software Engineering I

Lecture 4: Flow Control in C Programming

Dr. Rui Lin/Dr. Mark Leach
Office: EE512/EE510
Email: rui.lin/mark.leach@xjtlu.edu.cn
Dept. of EEE XJTLU

Outline of Today's Lecture (week 4)

- Making decisions - Branching
  - Choosing to do something if()
  - Choosing between two options if()…. else….
  - Choosing between several conditions if()…. else if()….
- Multiple choice menus
  - Switch
- Repeating operations – looping
  - while()
  - do…while()
  - for()

Assignment 1

- You should have by now submitted the first assignment and the marking process has started.
- Please remember, you should only question your marks if you think there has been a significant error.
- Do not expect to achieve 100% simply because your programme works...there is more to it!
- Check the marking guidelines

The option to choose…or not!

- **How do you get a program to follow some particular function and not others?**

The option to choose…or not!

- **How do you get a program to follow some particular function and not others?**

- The answer is of course **flow control**.

- In general, a programming language must provide 3 types:

  – Ordered processing of statements (**Sequential execution)**

  – Use of a test to decide between alternative sequences (branching/**conditional execution**)

  – Repeating sequences of statements until a condition is met (looping/**iterative execution**)

Flow Control Statements

- These enable you, the programmer, to make complex decision making trees, in a program

- They provide a choice of action or repetition of action.

- They require a test expression to be evaluated.

- Test expressions can have **two** values…
What do you think these are?

Flow Control Statements

- These enable you, the programmer, to make complex decision making trees, in a program
- They provide a choice of action or repetition of action.
- They require a test expression to be evaluated.
- Test expressions can have **two** values...
- **Unsurprisingly they can be true or false:**
  0 – Means False
  1 (or nonzero) – Means True
Test expressions commonly use **relational** or **logical** operations.

Statements and Blocks

- An **expression** in your code is something that ends in a semicolon ';'. For example x=0;

- **Do we need a semicolon after squiggly brackets {}?**

Statements and Blocks

- An **expression** in your code is something that ends in a semicolon ';'. For example x=0;

- **Do we need a semicolon after squiggly brackets {}?**

- **No, these brackets denote a functional block or a "compound statement".**

- **Example main() {....}   No semicolon**

- **if, else, while, for ... {} we will see the brackets are used to group multiple statements together.**

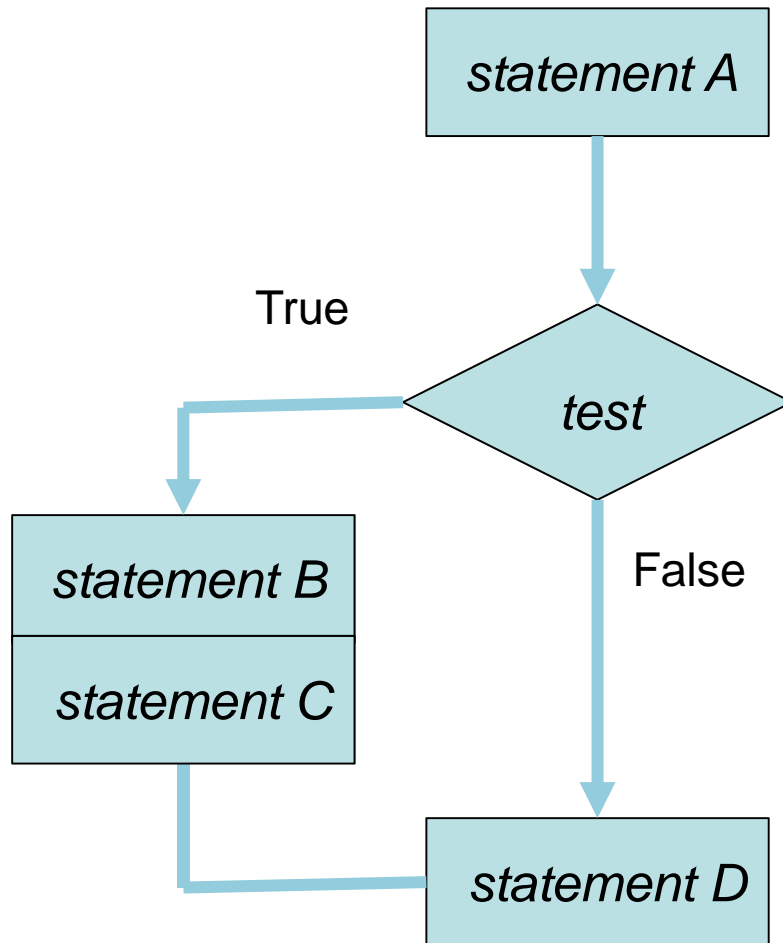# Choices...the **if**() statement

Choosing **to execute** a statement or **not**
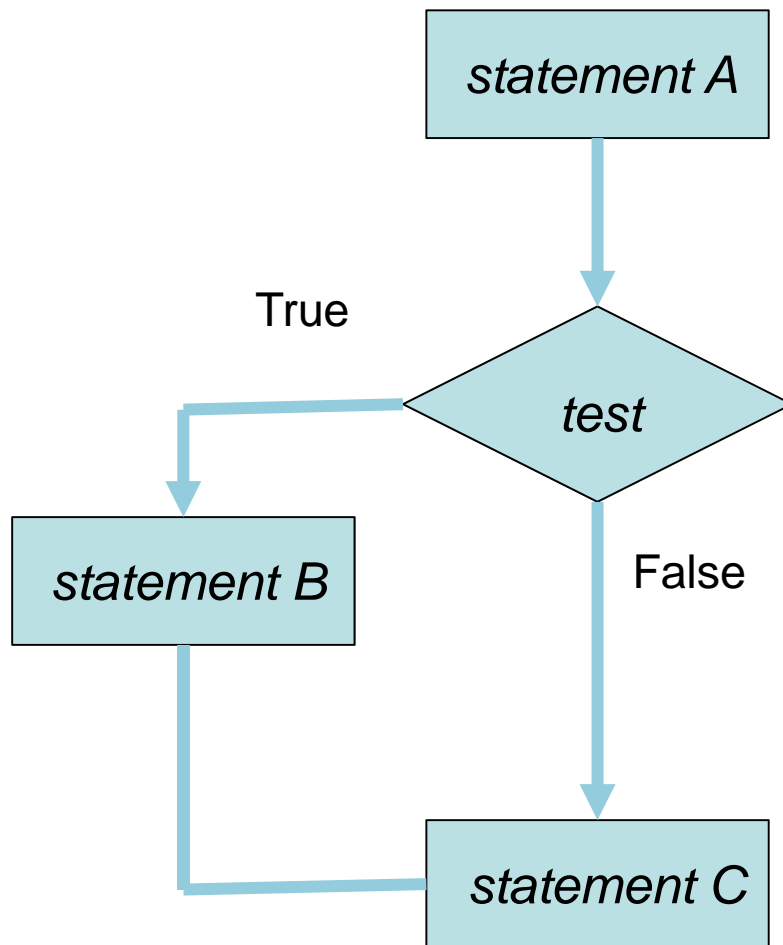

```
statement A;
if (test)
    statement B;
statement C;
```
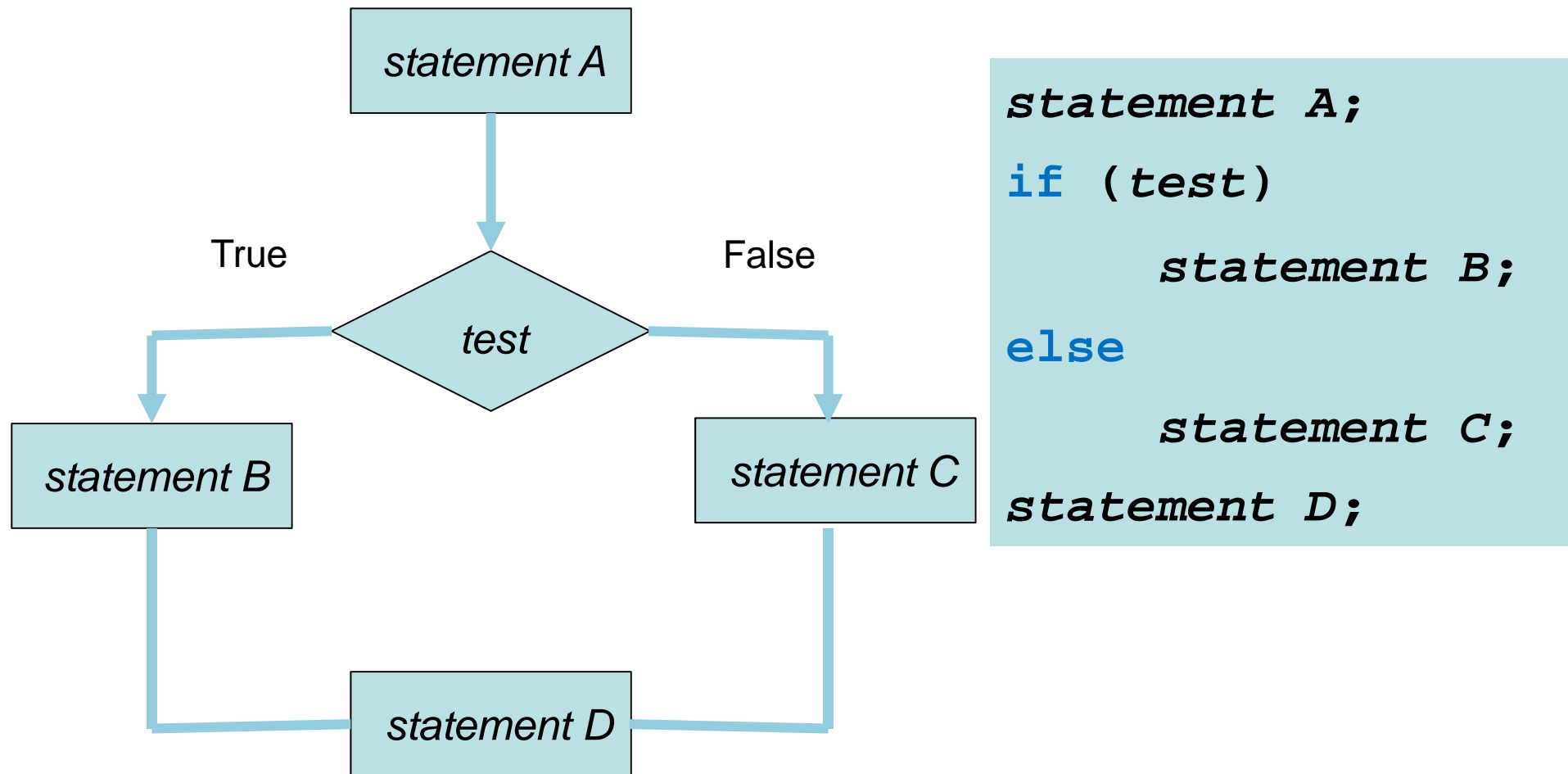
Choices...the **if**() statement

Choosing **to execute** a BLOCK of statements or **not**


```
statement A;

if (test)

{

    statement B;

    statement C;

}

statement D;
```
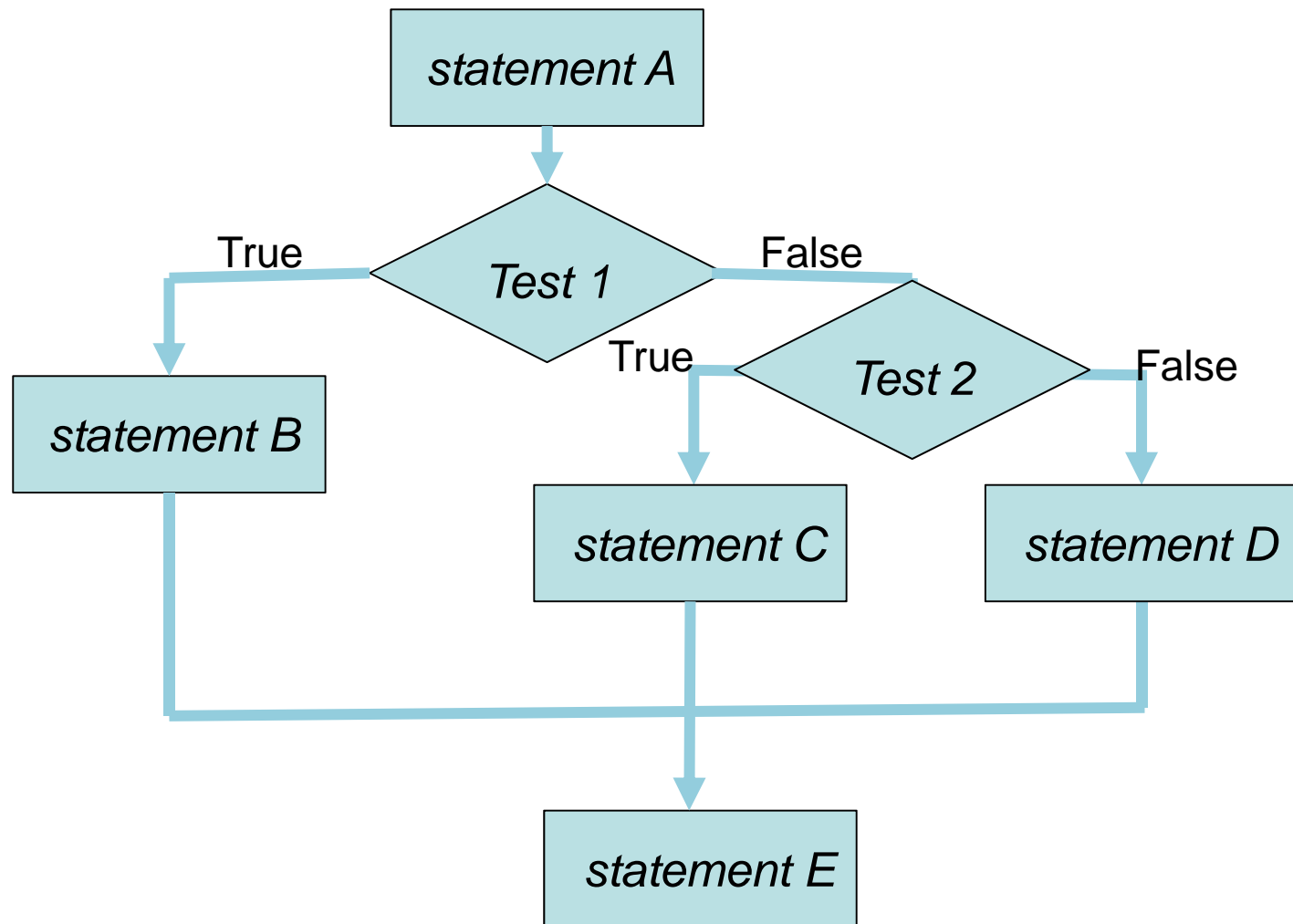
Choices…the **if**() **else** statement

Choosing **between two statements or blocks**

```
statement A;
if (test)
     statement B;
else
     statement C;
statement D;
```

statement A

True   False

test

statement B   statement C

statement D

the **if**() **else if**() ... statements

Choosing **between two statements or blocks**


```
statement A;
if (test1)
    statement B;
else
    if (test2)
        statement C;
    else

        statement D;
Statement E;
```

Pairs of **if**() and **else**

```
if ( number > 6 )
   if ( number < 12 )
  printf("You are close!");
else
    printf("You Lose!");
```

How does this work? (which if does else belong to)?
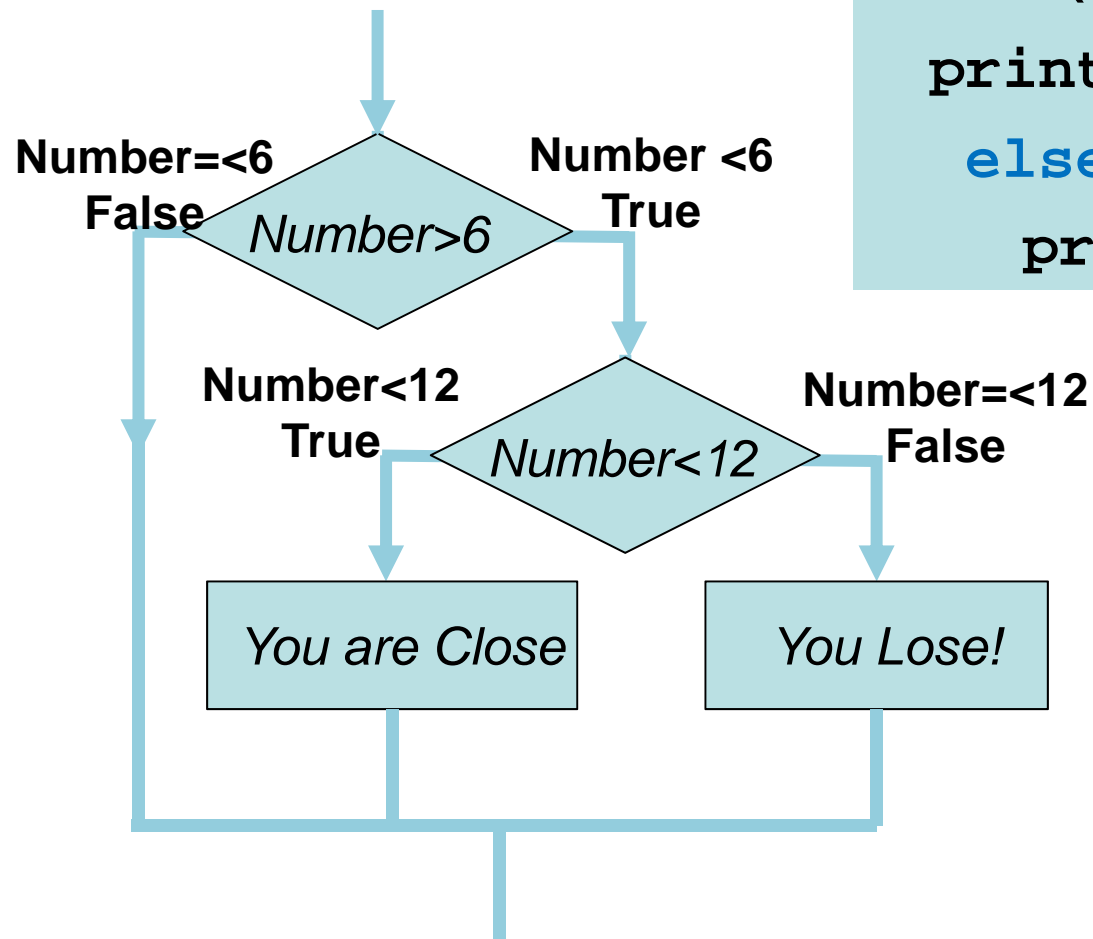
Pairs of **if**() and **else**

```
if ( number > 6 )
  if ( number < 12 )
    printf("You are close!");
else
    printf("You Lose!");
```

else pairs with the most recent **if** unless braces are used

`number`	output
5	none
7	You are close!
14	You loose!

Pairs of **if**() and **else**…flow chart

```c
if ( number > 6 )
  if ( number < 12 )
 printf("You are close!");
  else
    printf("You Lose!");
```


Number=<6
False

Number <6
True

Number>6

Number<12
True

Number=<12
False

Number<12

You are Close

You Lose!

Quick Quiz 1

What is the value of z?

```
int x = 5, y = 2, z = 0;
if(x>3)
if(y>2)
z=1;
else
if(x<10)
z=2;
else
z=3;
```

a) 0

b) 1

c) 2

d) 3

Quick Quiz 1

What is the value of z?

```
int x = 5, y = 2, z = 0;
if(x>3)
        if(y>2)
                z=1;
        else if(x<10)
                z=2;
        else
                z=3;
```

a) 0

b) 1

c) <u>2</u>

d) 3

Logic

The Grim Reasoner

- Logic is all about reasoning with statements

- Statements can be either true or false

- Logic can be used to deduce new or simpler statements

Logic and Common Sense

- So far, we have taken a procedural view of logic:

  if (x is true) do y
  else   do z

- Common sense tells us we can re-write this as:

  if (x is false) do z
  else   do y

But sometimes, common sense isn't reliable…

If your logic is wrong, the program will do the wrong thing!

Negating Logic – De Morgans Rules

Negation rule can be proved using De-Morgans Rules

P	Q	P && Q	!(P && Q)	!P	!Q	!P || !Q
T	T	T	F	F	F	F
T	F	F	T	F	T	T
F	T	F	T	T	F	T
F	F	F	T	T	T	T

$!(P \,\&\&\, Q) \equiv !P \,||\, !Q$

- Similarly, it can be shown that: $!(P \,||\, Q) \equiv !P \,\&\&\, !Q$

Simplifying C Conditionals

- When programming, it is usually easier to understand what conditional statements will do if they are expressed in a **positive** sense

- Example: Use De-Morgans rules to re-write

```
if(!(w<50 || h<50))
if(!(P || Q))          /*subs P = w<50,Q=h<50*/
if(!P && !Q)           /*apply DeMorgan's*/
if(w>=50 && h>=50)     /*subs !P = w>=50,etc*/
```

Quick Quiz 2

- **Simplify the following:**

```
if (!(p>=200 && p<0) && q>0)
```

```
1) if (p<200 && p>=0 && q>0)
2) if (p<200 || (p>=0 || q<=0))
3) if ((p<200 || p<0) || q<=0)
4) if (p>=0 || p<200 && q>0)
5) if ((p>=0 || p<200) && q>0)
```

Quick Quiz 2

- **Simplify the following:**

```
if (!(p>=200 && p<0) && q>0)
```

```
1) if (p<200 && p>=0 && q>0)
2) if (p<200 || (p>=0 || q<=0))
3) if ((p<200 || p<0) || q<=0)
4) if (p>=0 || p<200 && q>0)
5) if ((p>=0 || p<200) && q>0)
```

More Shorthand (Laziness!)

- **if else** has a shorthand form

Test first   If FALSE: `x=y`

```
x = (y < 0) ? -y : y;
```

If TRUE: `x=-y`

This is equivalent to

```
if ( y < 0 )
    x = -y;
else
    x = y;
```

test expression ? expression2 : expression3

Multiple Choice...**switch** and **break**

- **switch** is used to provide multiple choices

```
switch(integer expression)
{
    case constant1: statement1;
                    break;
    case constant2 : statement2;
                     break;
    default : statement3;
}
statement4;
```

The integer expression is evaluated, and (say) the result is `constant2`

This means that `statement2` will be executed next

The **break** statement makes the program **exit** the **switch** block and jump to `statement4`

- **switch** test expression must have **integer** value
- **case** labels must be **integer-type**
   (constant, variable or expressions)

switch without break

- **break** ends the code block. It is good practice to include it in all cases…including the default.

```
switch(integer expression)
{
case constant1 : statement1;
case constant2 : statement2;
case constant3 : statement3;
default        : statement4;
}
statement5;
```

The integer expression is evaluated and the result is *constant2*

This means that *statement2* will be executed next…

…then *statement3* is executed next…

… then *statement4* is executed next

switch and break example

```c
#include <stdio.h>
#include <simpio.h> //
GetInteger
main()
{
  int day;
  day = GetInteger();

  switch(day)
  {
    case 0 : printf("Sunday");
             break;
    case 6 : printf("Saturday");
             break;
    default : printf("Weekday");
             break;
  }
}
```

Read integer value - store it in **day**

If **day** is 0 print **Sunday**

If **day** is 6 print **Saturday**

The **break** statements make the program **exit** the **switch** block

For any other number print **Weekday**

Looping…the **while** loop

- In C, the syntax for a **while** loop looks like an **if** statement without **else**.

```
while (whileCondition) {
  /* do something */
}
```

- The body of the while statement is executed **repeatedly** as long as the condition is true.

the **while** loop

- Conditional loop with an entry condition

```
while (test)
    Statement A;
Statement B;
```

Check if test expression is TRUE

If the expression is TRUE execute `Statement A`

Check again if the test expression is TRUE

The loop is exited **ONLY** if the test expression is FALSE

The **while** loop

```
while(test)
{
  statement1;
  statement2;
  . . .
}
int i=0,n,sum=0;
long factorial=1;
scanf("%d",&n);
while(i<n)
{     i = i+1;
     factorial *= i;
     sum += i;
}
```

If the test expression is TRUE execute the **loop body** (the block of statements in between the set of curly brackets **{ }**)

- Note that part of the loop body relates to the loop condition

- It is **good style** to indent the loop body statements

The **do while** loop

What if you want to make sure the loop content executes once, whatever the test result?

The **do while** loop

What if you want to make sure the loop content executes once, whatever the test result?

The **do while** loop allows for this (**while** becomes an **exit** condition)

```
do
    statement;
while(test);
```

Use the test to decide if you want to have another go … execute the **statement** or **block of statements** again

```
count = 1;
do
{
    prinf("La,la,la!");
    count ++;
}
while(count < 5);
```

The do while or while loop?

Decide if you need an **entry** or an **exit** condition

Entry condition loops are preferred

Better to look before leaping

Entry condition loops make programs easier to read

Use **for** when the loop requires initialisation and update a variable for control.

while is ideal for conditions like:

```
while ( scanf("%d", &num) != 1 )
```

Quick Quiz 3

What number is printed after the loop?

```c
float fac = 1.0;
int n = 0;
while (1){
   n++;
   if (fac*n > 50) break;
   fac = fac * n;
}
printf("Factorial = %f ", fac);
```

a) 1

b) 2

c) 6

d) 24

e) 120

Quick Quiz 3

What number is printed after the loop?

```c
float fac = 1.0;
int n = 0;
while (1){
   n++;
   if (fac*n > 50) break;
   fac = fac * n;
}
printf("Factorial = %f ", fac);
```

a) 1

b) 2

c) 6

d) <u>24</u>

e) 120

The **for** Loop

- Combines 3 actions in one place:

```
for(init;test;step)
        statement;
next statement;
```

```
for( i=1;i<=10;i++ )
    factorial*=i;
printf("%d",factorial);
```

- *init* – **initialise counter variable**

- *test* – **logical condition**

- *step* – **modify counter variable**

The **for** Loop

- Combines 3 actions in one place:

```
for(init;test;step)
        statement;
next statement;
```

```
for( i=1;i<=10;i++ )
        factorial*=i;
printf("%d",factorial);
```

Equivalent to

```
init;
while(test)
{
        statement;
        step;
}
next statement;
```

```
i=1;
while( i<=10)
{
        factorial*=i;
        i++;
}
printf("%d",factorial);
```

Quick Quiz 4

What number is printed after the loop?

```
int n, total=0;
for (n=1; n<5; n++)
  total = total + n*n;

printf("Total: %d\n", total);
```

a) 5

b) 10

c) 16

d) 25

e) 30

Quick Quiz 4

What number is printed after the loop?

```c
int n, total=0;
for (n=1; n<5; n++)
  total = total + n*n;

printf("Total: %d\n", total);
```

a) 5

b) 10

c) 16

d) 25

e) <u>30</u>

Indefinite (**while**) vs. Counting (**for**)

- Indefinite loops – do not know in advance how many times the loop will be executed.

- Counting loops – the loop is executed a fixed (known) number of times.

- **A counter should be initialised**

- **The counter is compared with a limiting value**

- **The counter is incremented each time the loop is completed.**

The **for** Loop

```c
#include <stdio.h>
main()
{
  double debt;

  for (debt=100.0; debt< 150;debt*=1.1)

  printf ("Your debt is now %.2f\n", debt);
}
```

This time the **update expression** involves some extra arithmetic calculations.

Program output

```
Your debt is now 100.00
Your debt is now 110.00
Your debt is now 121.00
Your debt is now 133.10
Your debt is now 146.41
```

The **for** Loop…counting down

In this example our counting variable is **decremented** every time the loop is executed

```c
#include <stdio.h>
main()
{
  int secs;
  for(secs=5;secs>0;secs--)
    printf("%d seconds\n",secs);
  printf("We have ignition!");
  return 0;
}
```

Program output

```
5 seconds
4 seconds
3 seconds
2 seconds
1 seconds
We have ignition!
```

The **for** Loop...increment options

In this example our counter variable is **increased by 12** every time the loop is executed

```c
#include <stdio.h>
main()
{
    int n;
    for(n=0;n<55;n=n+12)
        printf("%d \n", n);

}
```

Program output

```
0
12
24
48
```

The **for** Loop…by character

Remember that characters are just numbers

ASCII code . . .

```c
#include <stdio.h>
main()
{
  char ch;
  for (ch='a'; ch<='z';ch++)
    printf("ASCII value of %c is %d\n", ch, ch);
}
```

Here we compare and increment the numeric code for the character stored in `ch` variable

The **for** Loop…changing conditions

```c
#include <stdio.h>
main()
{
  int n;
  for (n=0;n*n < 12; ++n)
    printf ("%d \n", n);
}
```

Our **test expression** involves some extra arithmetic calculations. Do you remember which operator is **evaluated first** – has the **higher precedence?**

Program output

```
0

1

2

3
```

The **for** Loop…omitting expressions

- Any of the 3 expressions in the **for** statement can be left out.
- **If expression 1 (init) or 3 (step) are omitted then they just don't happen**
- **If expression 2 (test) is omitted then the loop appears permanently true.**

Here we have omitted the expression that **tests** the condition, **the loop will never be terminated!**
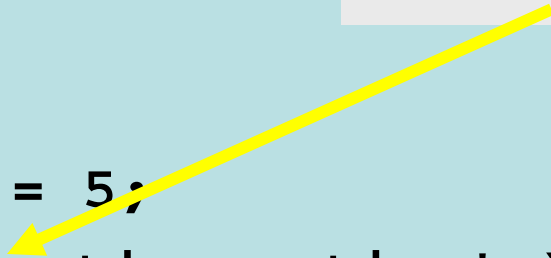
```c
#include <stdio.h>
main()
{
  int time = 5;
  for (n=2;; time = time*n)
   printf("n=%d;time is %d.",n,time);
}
```

The **for** Loop…nested loops

- A loop inside another loop

- **Used to display data as rows and columns**

```c
#include <stdio.h>
#define ROWS 4
#define CHARS 4

main ()
{
  int row; char ch;
  for (row = 0;row <ROWS; row++)
  {
   for (ch=('A'+row);ch<('A'+CHARS);ch++)
     printf ("%c",ch);
  printf("\n");
  }
}
```

Program output

```
ABCD

BCD

CD

D
```

Mid-Test Loops (1/2)

- Sometimes it is useful to use an infinite loop and make the termination test inside the loop using **break** or **continue**

- Try not to use **goto** or **label**, it is poor practice

- Example: find the first prime number>1000000

```
int p = 1000000;
while (1) {              /* could also use: for (; ;) */
                        /* 1st attempt is 1000001 */
  p++;
  if (isPrime(p))
    break;              /* details of isPrime is not shown */
                        /* break jumps here when isPrime(p) is true */
}
```

Mid-Test Loops (2/2)

- **break** – exit the loop at immediately (what about nested loops)
- **Continue** – skip the rest of the current cycle and start a new iteration

```
for(i=1;i<=10;i++)
{
    ch = getchar();
    if ( ch == '\n' )
        continue;
    putchar(ch);
}
```

```
for(i=1;i<=10;i++)
{
    ch = getchar();
    if ( ch == 'q' )
        break;
    putchar(ch);
}
```

Quick Quiz 5

The loop should print all numbers between 1 and 100. What is wrong with it?

```
int k;
for (k=1; k!=100; k=k+2)
{    printf ("Odd: %d \n", k);
     printf ("Even: %d \n", (k+1));
}
```

a) k=1;
b) k!=100
c) k=k+2
d) "Odd: %d \n",
e) (k+1)

Quick Quiz 5

The loop should print all numbers between 1 and 100. What is wrong with it?

```c
int k;
for (k=1; k!=100; k=k+2)
{    printf ("Odd: %d \n", k);
     printf ("Even: %d \n", (k+1));
}
```

a) k=1;
b) k!=100
c) k=k+2
d) "Odd: %d \n",
e) (k+1)

```c
/* Balances a Checkbook */
#include
<stdio.h> int
main()
{
  int cmd;
  float balance=0, credit, debit;
  printf("*** ACME checkbook-balancing program *** \n");
  printf("Commands: 0=clear, 1=credit, 2=debit, ");
  printf("3=balance, 4=exit \n\n");
  for(;;)
  {
    printf("Enter command: ");
    scanf("%d",&cmd);
    switch (cmd)
    { case 0:
        balance =
        0.0; break;
      case 1:
        printf("Enter amount of credit: ");
        scanf("%f", &credit);
        balance +=
        credit; break;
      case 2:
        printf("Enter amount of debit: ");
        scanf("%f", &debit);
        balance -=
        debit; break;
      case 3:
        printf("Current balance: $%.2f\n", balance);
        break;
      case 4:
        return
        0;
      default:
        printf("\nCommands: 0=clear, 1=credit, 2=debit, ");
        printf("3=balance, 4=exit \n\n");
        break;
    } /* end of switch   */
  } /* end of for */
} /* end of main function */
```

a program to show how to use selection statement and iteration statement

to realize a menu for a basic management information system

As always...
Thank you for your attention

See you in the lab sessions ☺