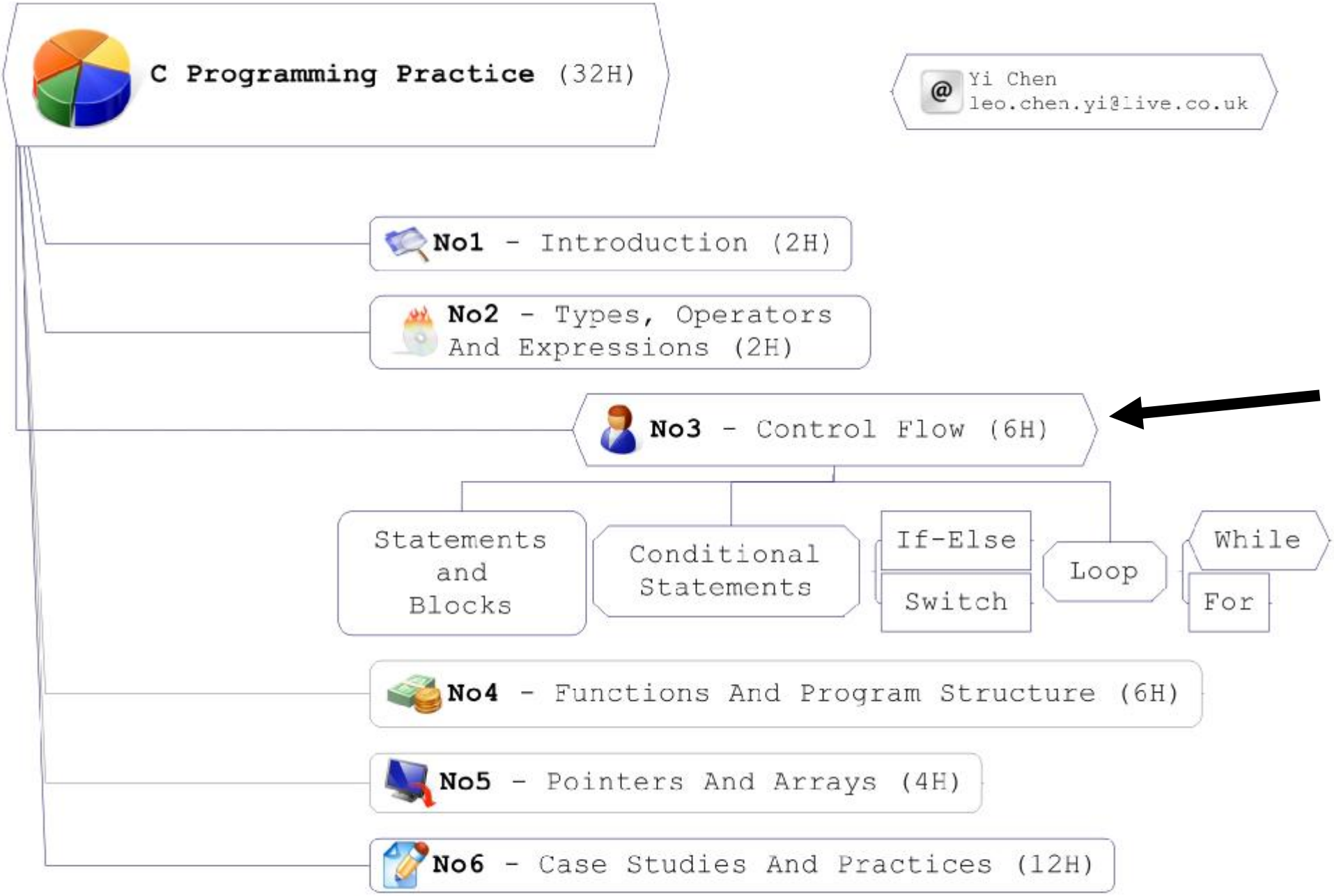
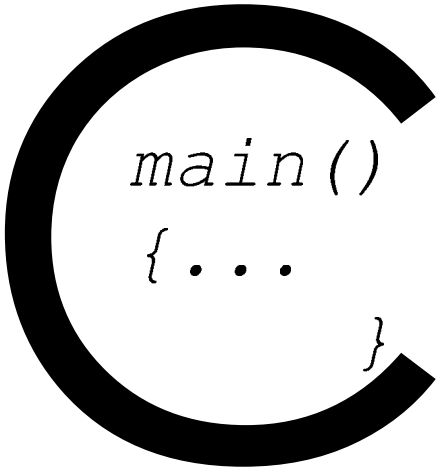

C Programming Practice (32H)
Yi Chen
leo.chen.yi@live.co.uk
No1 - Introduction (2H)
No2 - Types, Operators And Expressions (2H)
No3 - Control Flow (6H)
Statements and Blocks
Conditional Statements
If-Else
Switch
Loop
While
For
No4 - Functions And Program Structure (6H)
No5 - Pointers And Arrays (4H)
No6 - Case Studies And Practices (12H)

C Programming Practice No[3]

Chen , Yi
leo.chen.yi@live.co.uk
30-Jul-2010



Control Flow

BETA 1.0.0.1

Contents

Statements and Blocks
If-Else
Else-If
Switch
Loops - While and For
Loops - Do-While
Break and Continue
Goto and labels

Contents

à Statements and Blocks

If-Else

Else-If

Switch

Loops - While and For

Loops - Do-While

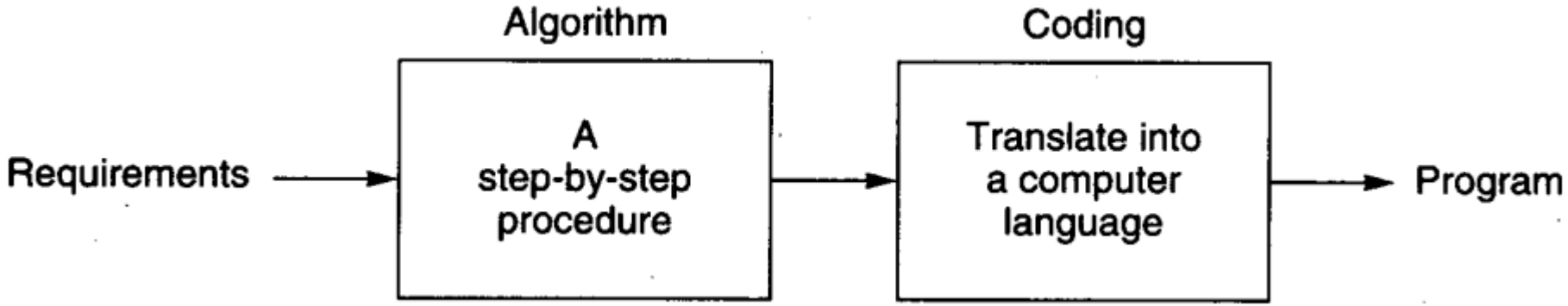
Break and Continue

Goto and labels

Algorithm

Algorithm:
the **step-by-step sequence** of instructions that describe **how** the data is to be **processed** to produce the desired output

Programming = the translation of the **selected algorithm** into a language the computer can use



Algorithm

Example: Calculate the sum of all whole numbers from 1 through 100

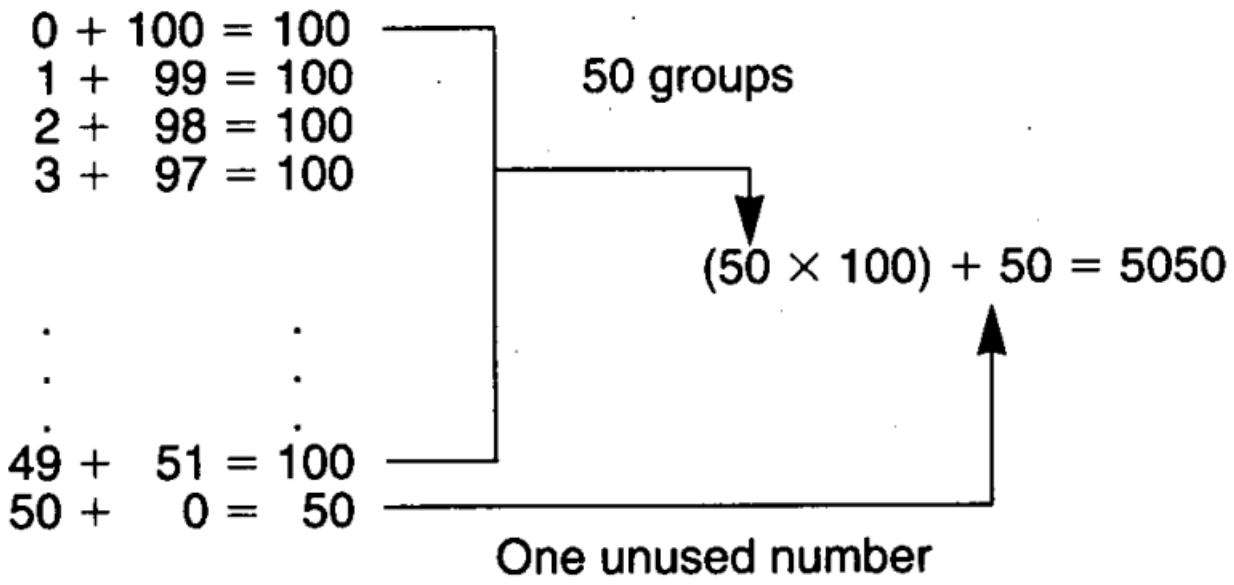
Method 1. *Columns:* Arrange the numbers from 1 to 100 in a column and add them:

$$
\begin{array}{r}
1 \\
2 \\
3 \\
4 \\
\vdots \\
98 \\
99 \\
+100 \\
\hline
5050
\end{array}
$$

Algorithm

Example: Calculate the sum of all whole numbers from 1 through 100

Method 2. *Groups:* Arrange the numbers in convenient groups that sum to 100. Multiply the number of groups by 100 and add in any unused numbers:



Algorithm

Example: Calculate the sum of all whole numbers from 1 through 100

Method 3. *Formula:* Use the Formula

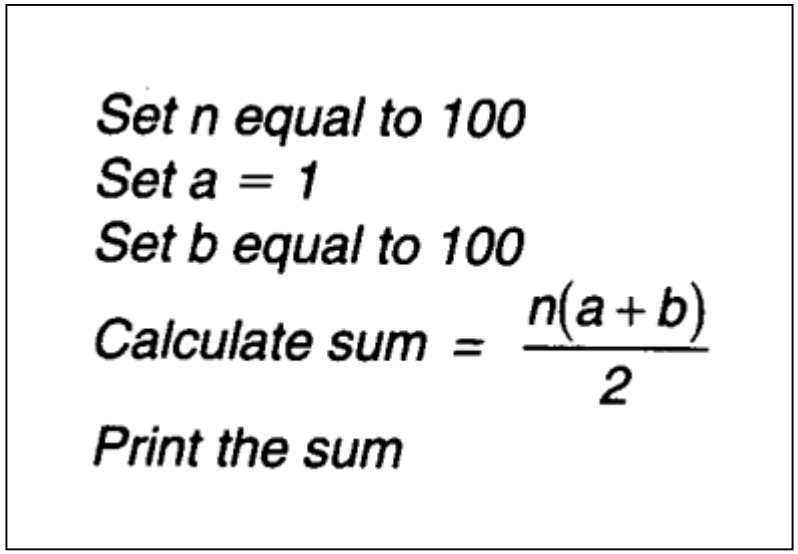
$$\text{Sum} = \frac{n(a + b)}{2}$$

where

n = number of terms to be added (100)
a = first number to be added (1)
b = last number to be added (100)

$$\text{Sum} = \frac{100\ (1 + 100)}{2} = 5050$$

Algorithm

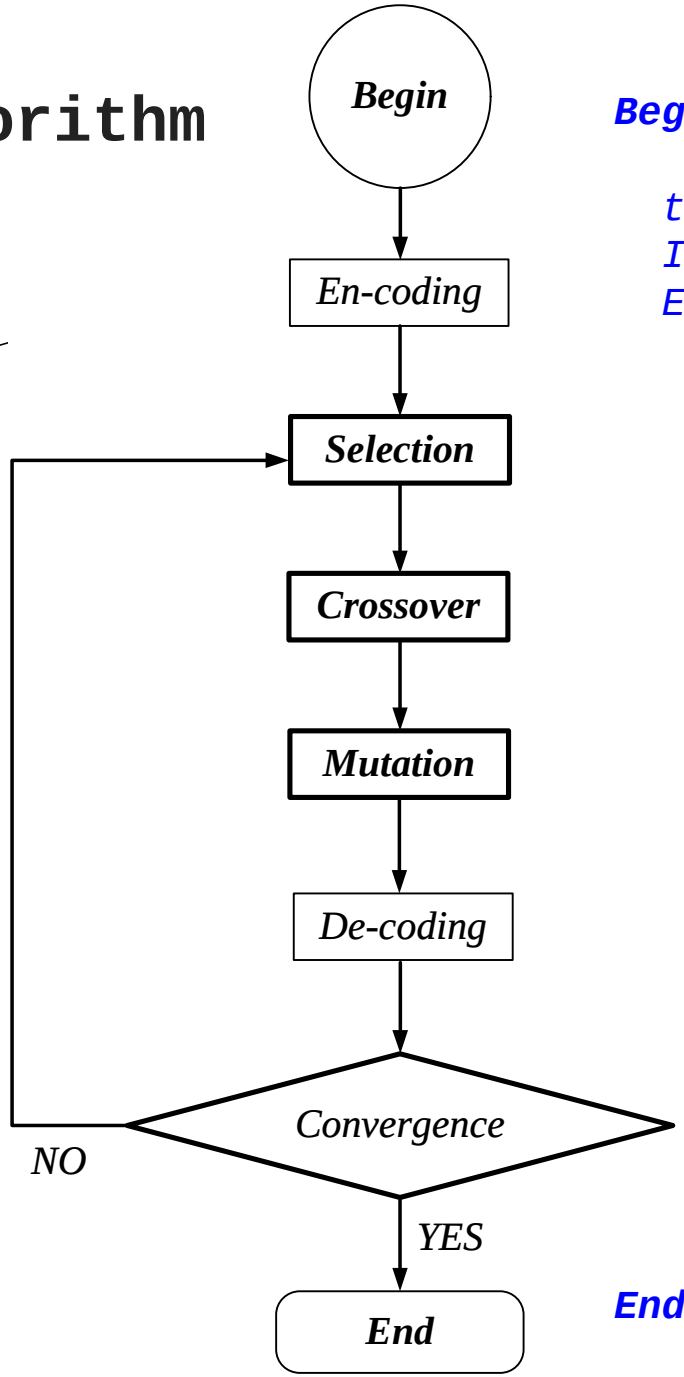


Pseudocode: English-like phrases used to describe the algorithm

Formula: description of a mathematical equation

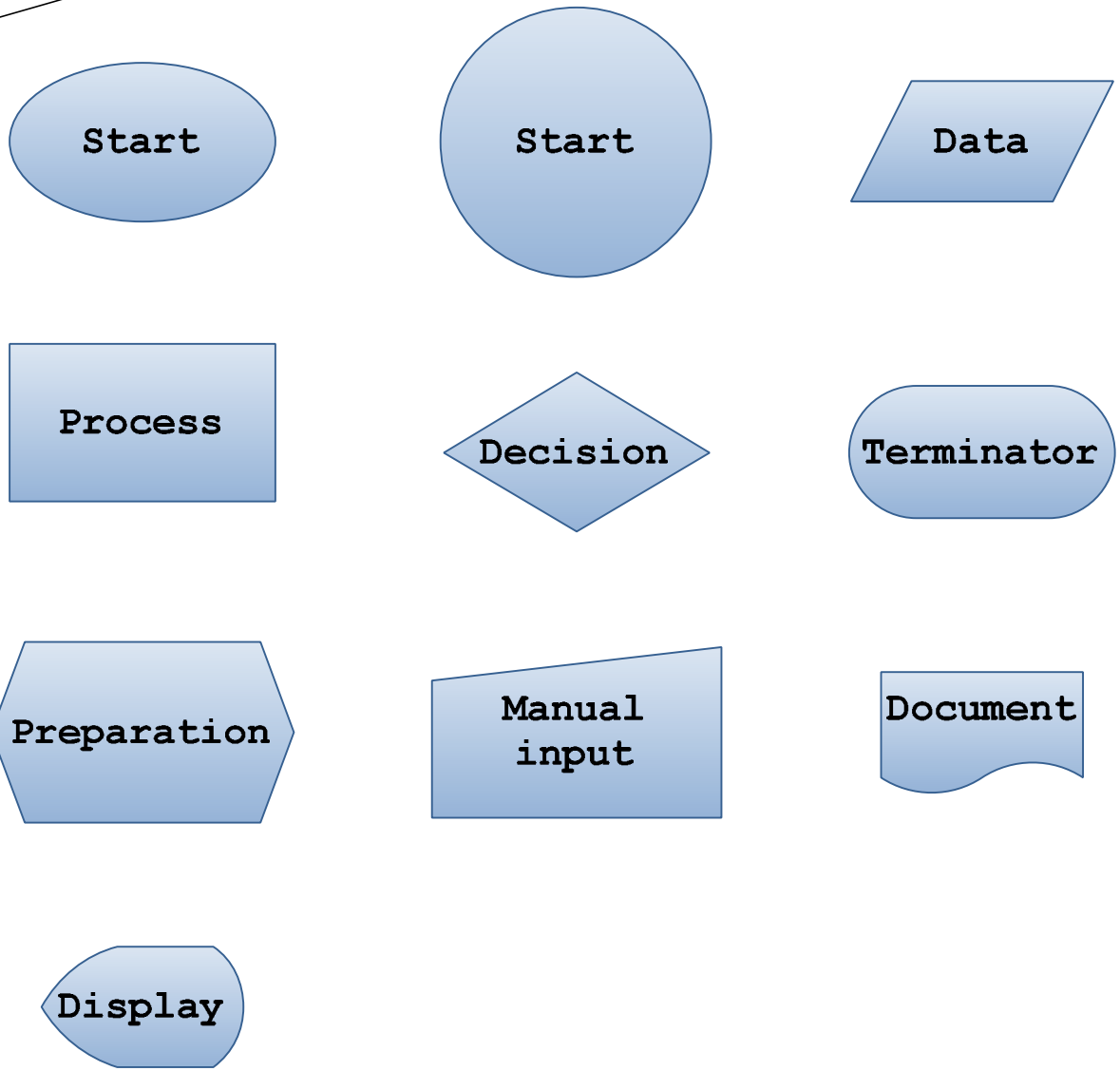
Flowchart: diagram showing the flow of instructions in an algorithm, Flowcharts use special symbols

Algorithm




```
Begin (1)

  t = 0 ;
  Initialise P(t);
  En-coding P(t);

    While ( Not termination-condition) do

      Begin (2)

        {
          t = t+1;

          Select P(t) from P(t-1)
          Crossover P(t);
          Mutation P(t);

          De-coding P(t);
          Evaluate P(t);
                                    }

      End (2)

End (1)
```

Flowchart Shapes



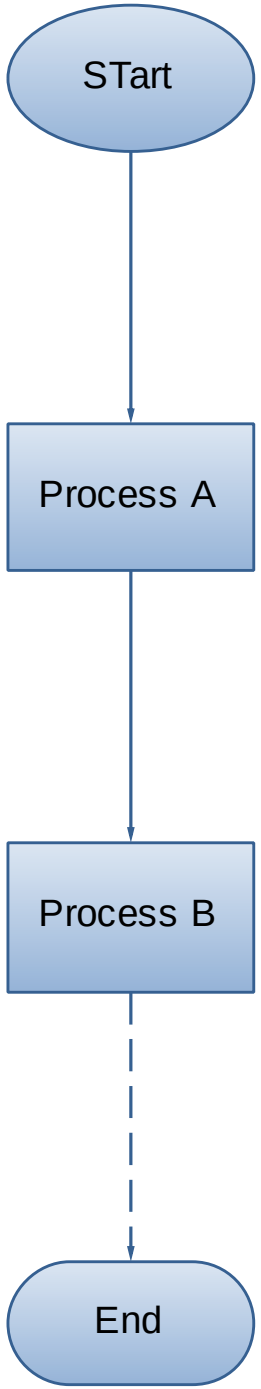
Statements and Blocks

An **expression** such as x = 0 or i++ or printf(...) becomes a *statement* when it is followed by a **semicolon(;)**, e.g. x = 0; i++; printf(...);

In C, the **semicolon** is a statement terminator, rather than a separator as it is in languages like Pascal.

Braces **{** and **}** are used to group declarations and statements together into a ***compound statement***, or ***block***, so that they are syntactically equivalent to a single statement.

No semicolon at a **block end**



Statements and Blocks

> a simple statement ends in a semicolon:

```
days = number * weeks;
```

> the multiple statements:

```
{
  temp  =  xdata + ydata ;

  zdata  =  foo ( temp ) ;
                          }
```

> Variables can be declared inside

```
{
    int  temp  =  xdata + ydata ;

         zdata =  foo ( temp )  ;
                                 }
```

Statements and Blocks

Blocks nested **inside** each other

```
{
   int temp = xdata + ydata;
   zdata = foo( temp ) ;

  {
     float temp2  = xdata * ydata ;

     zdata += bar( temp2 ) ;
                          }

                                }
```

Control conditions

The **if** statement

The **switch** statement

> Unlike C++ or Java, no boolean type (in C89/C90)

> in C99, bool type is available (use stdbool.h)

> Condition is an **expression** (or series of expressions)
e.g. degree < 45 or xdata < ydata || zdata < ydata

> Expression is **non-zero** condition true, the expression must be a numeric (or a pointer)

```
const char str [] = "some  text" ;

if  (str) /* string  is  not null */

return   0;
```

Contents

Statements and Blocks
à If-Else
Else-If
Switch
Loops - While and For
Loops - Do-While
Break and Continue
Goto and labels

If-Else

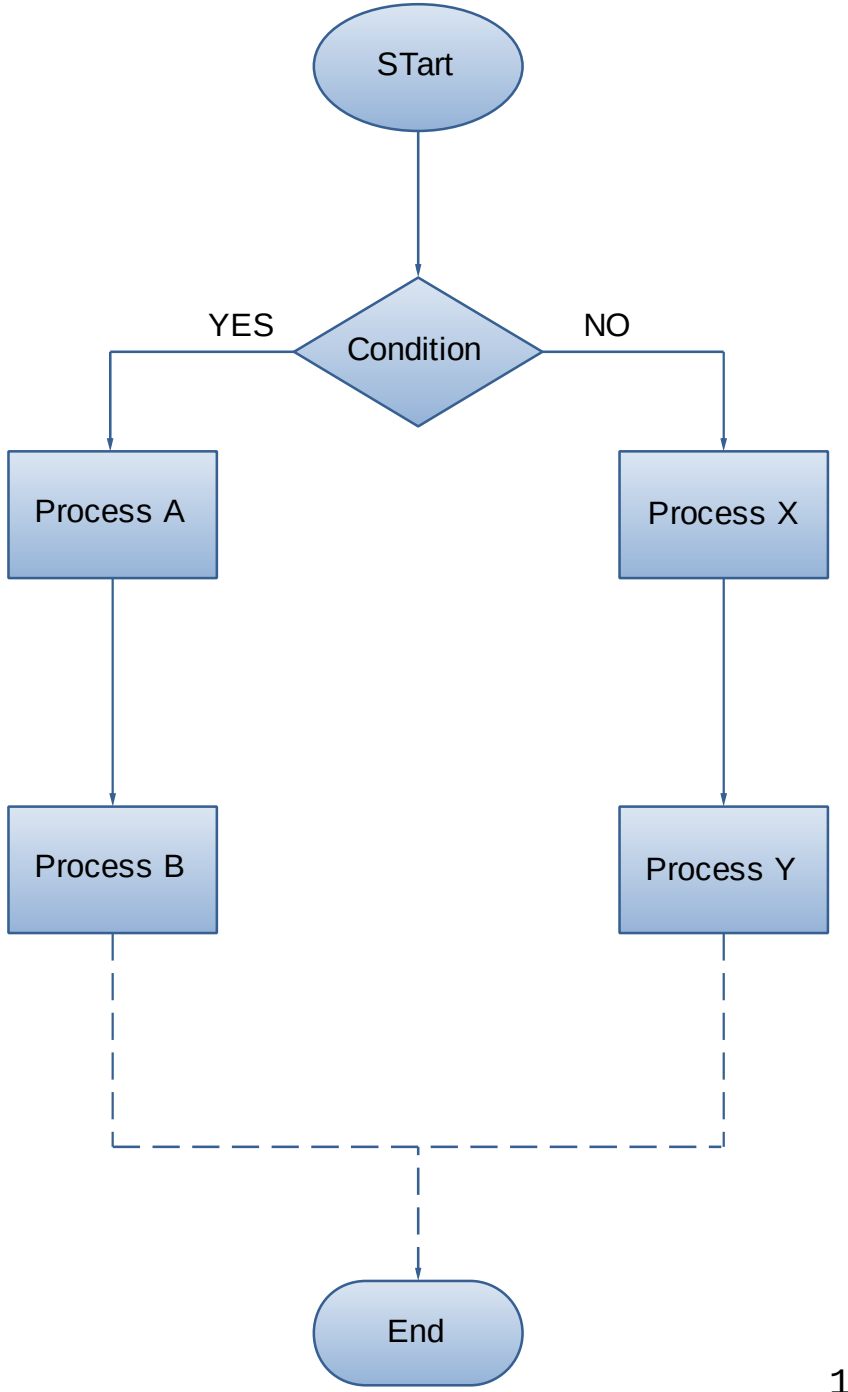
The if-else statement is used to express decisions. Formally the syntax is

```
if ( expression )
    statement1
else
    statement2
```

where the **else** part is optional, and the expression is evaluated; if it is **true** (that is, if expression has a non-zero value), **statement1** is executed. If it is **false** (expression is zero) and if there is an else part, **statement2** is executed instead.

If-Else

```
 if ( Condition )
{
        Process A
        Process B
                    }
 else
{
        Process X
        Process Y
                    }
```



If-Else

Because the **else** part of an if-else is **optional**, there is an ambiguity when an else if omitted from a nested if sequence.

```
if ( week%4 == 0 )
   if ( hour%2 == 0 )
      ydata = 2;
   else
      ydata  =  1;
```

=

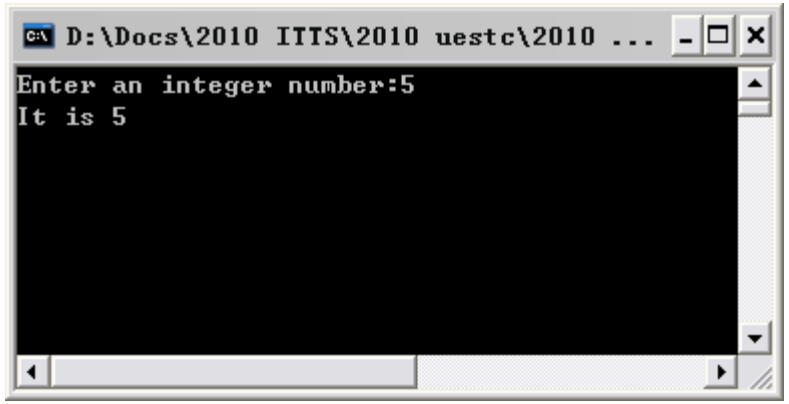
```
if ( week%4 == 0 )
{
   if ( hour%2 == 0 )
      ydata = 2;
   else
      ydata  =  1; }
```

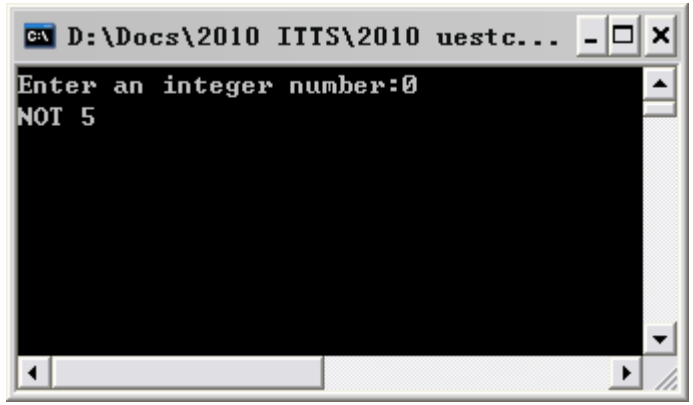
If-Else

To associate else with outer if statement:
use braces {}

```
if ( week%4 == 0 )
{
   if ( hour%2 == 0 )

      ydata = 2;   }

else

      ydata  =  1;
```

If-Else





cp_case_ifelse.c

```
#include <stdio.h>

int main( void )
{
  int    num = 0;

  printf("Enter an integer number:");

  scanf( "%d", &num);

  if ( num  ==   5 )
   {
       printf("It is 5");
       getch();
                      }
  else
   {
       printf("NOT 5");
       getch();
                      }
return 0;                                  }
```

Contents

Statements and Blocks
If-Else
à Else-If
Switch
Loops - While and For
Loops - Do-While
Break and Continue
Goto and labels

Else-If

This sequence of if statements is the most general way of writing a **multi-way decision**.

The expressions are evaluated **in order**; if an expression is true, the statement associated with it is executed, and this terminates the whole chain.

The construction:

```
if ( expression1 )
    statement1
else if ( expression2 )
    statement2
else if ( expression3 )
    statement3
else if ( expression4 )
    statement4
else
    statement5
```

Else-If

As always, the code for each statement is either a single statement, or a group of them **in braces**.

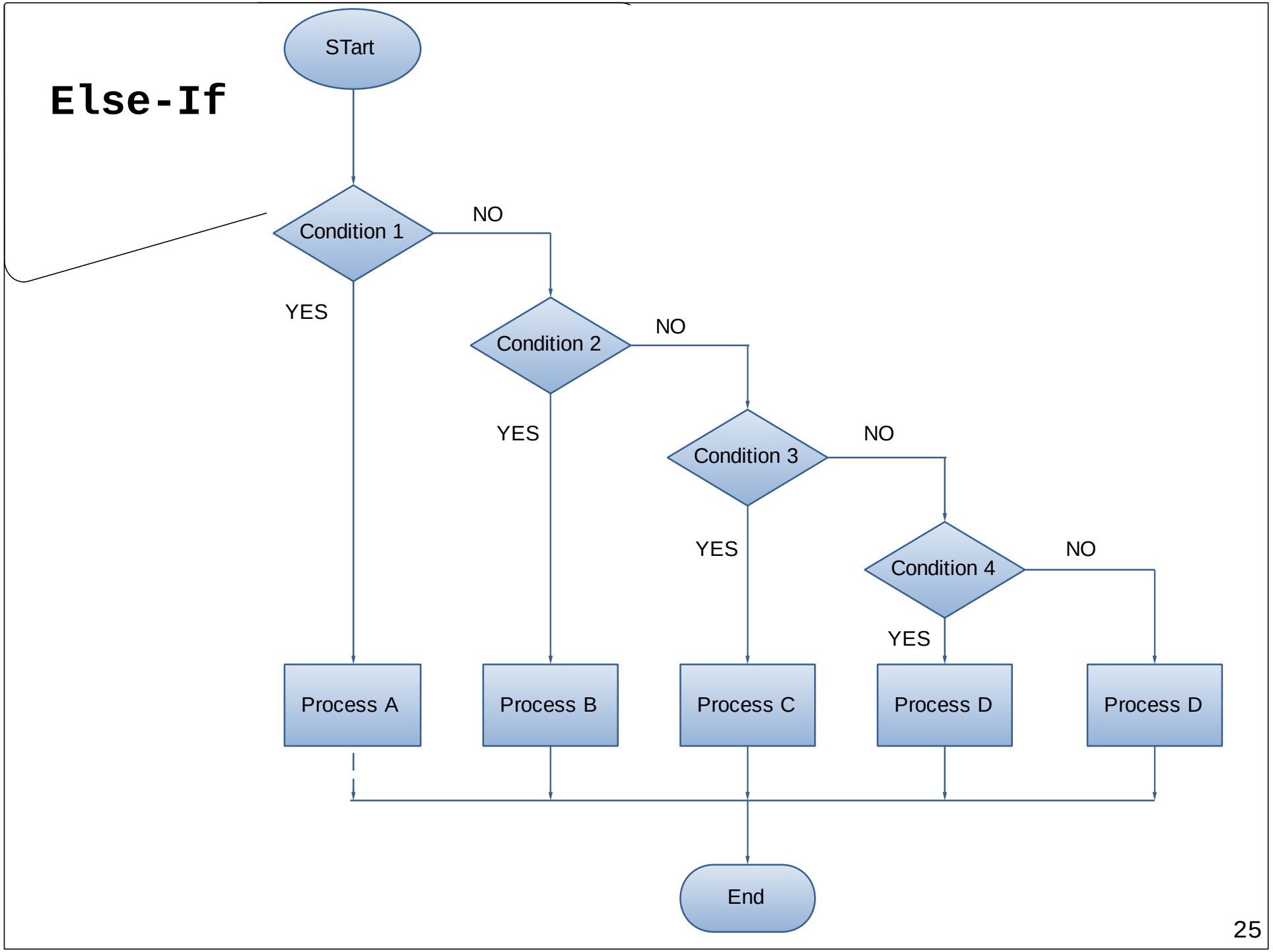
The **last else** part handles the ``**none of the above**'' or default case where none of the other conditions is satisfied.

Sometimes there is no explicit action for the default; in that case the trailing

```
else
    statement
```

can be omitted, or it may be used for **error checking** to catch an ``impossible'' condition.

Else-If

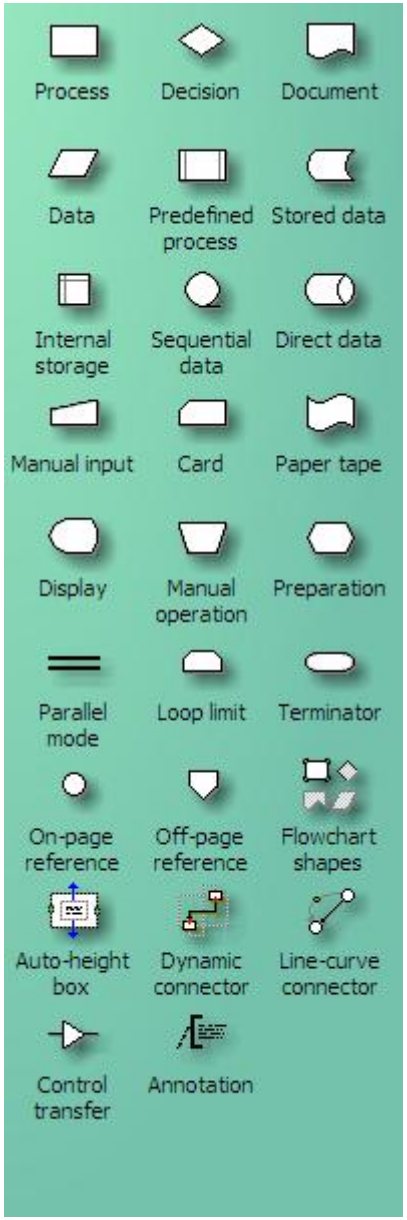

STart
Condition 1
NO
YES
Condition 2
NO
YES
Condition 3
NO
YES
Condition 4
NO
YES
Process A
Process B
Process C
Process D
Process D
End

Else-If

operator	meaning	examples
>	Greater than	3 > 2 /* 1 */ 2.99 > 3 /* 0 */
>=	Greater than or equal to	3 >= 2 /* 1 */ 2.99 >= 3 /* 0 */
<	Smaller than	3 < 2 /* 0 */ 2.99 < 3 /* 1 */
<=	Smaller than or equal to	3 <= 3 /* 1 */ 3.99 <= 3 /* 0 */
==	equal to	day == 7
! =	NOT equal to	day != 7

Contents

Statements and Blocks
If-Else
Else-If
à Switch
Loops - While and For
Loops - Do-While
Break and Continue
Goto and labels



Switch

The switch statement is a **multi-way** decision that tests whether an expression matches one of a number of *constant* integer values, and branches accordingly.

```
switch (expression)
{
 case const-expr1:
      Statements11;
      Statements12;
        …
      Break;
case const-expr2:
     Statements2
     Statements21;
     Statements22;
       …
     Break;

default:
     StatementsAA;
     StatementsBB;
       …
     Break;

}
```

Switch

Each case is labeled by **one** or **more** integer-valued constants or constant expressions.

If a case matches the expression value, execution starts at that case.

All case expressions must be **different**.

The case labeled **default** is executed if none of the other cases are satisfied.

A **default** is **optional**; if it isn't there and if none of the cases match, no action at all takes place.

Cases and the default clause can occur **in any order**.

Switch

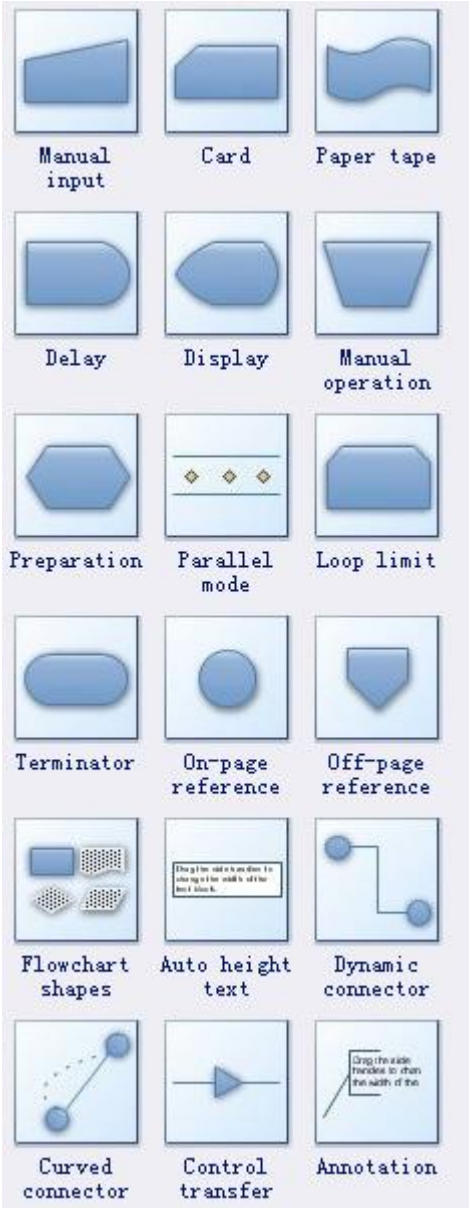
Integer (or **character**) variable as input

Execution “falls through” if **break**; not included

```
switch   ( ch )
{
case  'Y' :   /*   ch  ==   'Y '  */
              /*   do  something  */
    break ;

case  'N' :   /*    ch  ==   'N '  */
              /*    do  something   else  */
    break ;

default :   /*    otherwise  */
            /*    do a  third  thing */
    break ;
}
```

Contents

Statements and Blocks
If-Else
Else-If
Switch
à Loops - While and For
Loops - Do-While
Break and Continue
Goto and labels



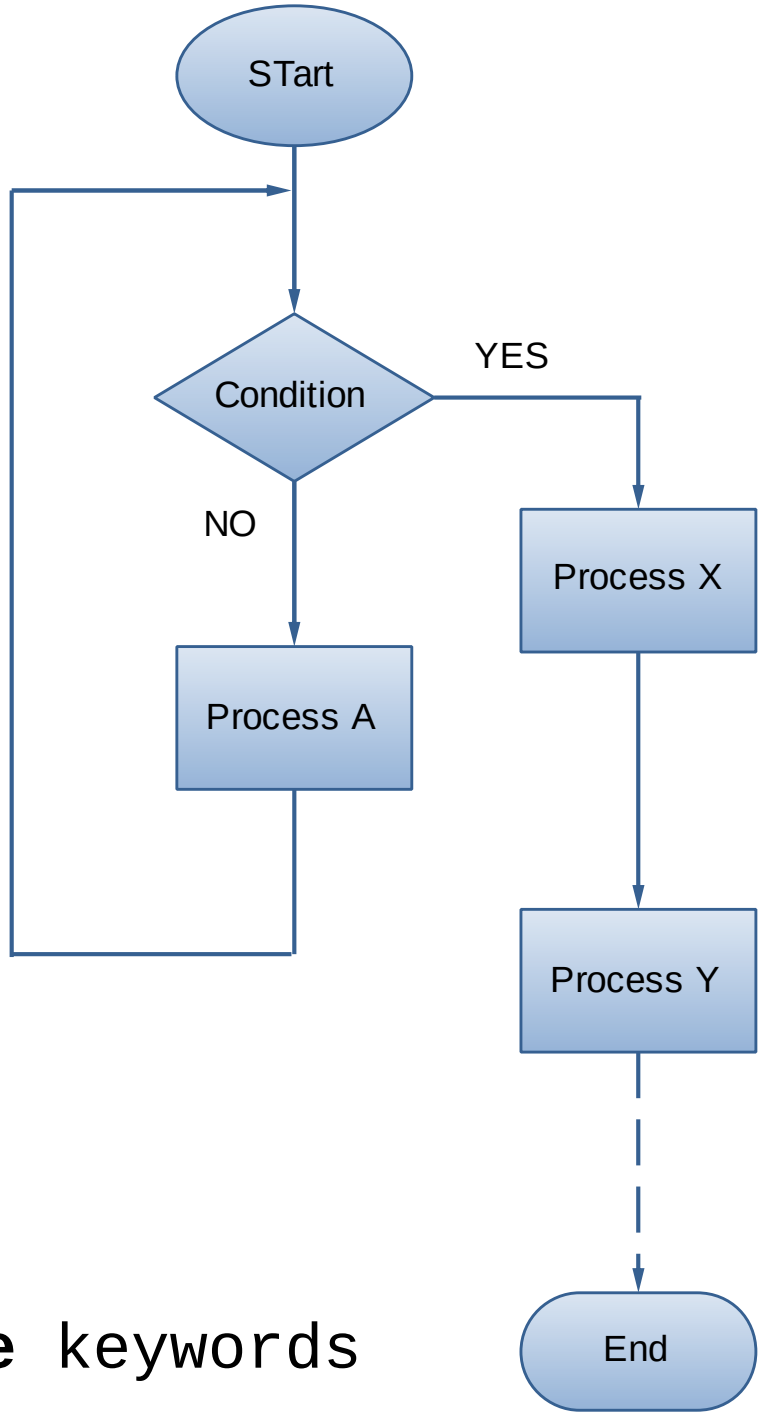
Loop statements

The **while** loop

The **for** loop

The **do-while** loop

The **break** and **continue** keywords



The while loop

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

The for loop

> The "**counting**" loop,
> given **total loop number**

```
int factorial ( int num )
 {
    int   ind = 0;
    int   jnd = 0;

for ( ind =1; ind <= num; ind++ )
      jnd *= ind ;

return   jnd ;     }
```

Inside parentheses, three expressions, separated by semicolons:

Initialisation: *ind = 0;*
Condition: *ind <= num*
Increment: *ind++*

Expressions can be **empty** (condition assumed to be "**true**")

The for and while loop

Equivalent to while loop:

```
int factorial_2 ( int num )
 {
    int   ind = 1;
    int   jnd = 1;

while ( ind <= num )
 {
     jnd *= ind ;

     ind++;
                  }
return   jnd ;
                        }
```

The for loop

Add 1 to 10, the step is 1

```
int sum  = 0;
int ind  = 0;
int num  = 10;
int step = 1;

for ( ind =1; ind <= num; ind++ )
 sum = sum + step ;
```

The for loop

Add 0 to 1, the step is 0.1

```
int sum  = 0;
int ind  = 0;
int num  = 10;
int step = 0.1;

for ( ind =1; ind <= num; ind++ )
 sum = sum + step ;
```

Contents

Statements and Blocks
If-Else
Else-If
Switch
Loops - While and For
à Loops - Do-While
Break and Continue
Goto and labels

Do-While

Differs from **while** loop – condition evaluated **after** each iteration

Body executed **at least** once
Note **semicolon** at end

```
char  accept = [] ;

do{
/* loop body */
puts ( "Keep  going?  (y/n)  " ) ;

accept =  getchar ( ) ;

/* other processing */
}
while ( accept ==  'y'  &&  /* other conditions */ ) ;
```

Contents

Statements and Blocks
If-Else
Else-If
Switch
Loops - While and For
Loops - Do-While
à Break and Continue
Goto and labels

Break

Sometimes want to **terminate** a loop **early**

break; exits **inner-most** loop or **switch** statement to exit early

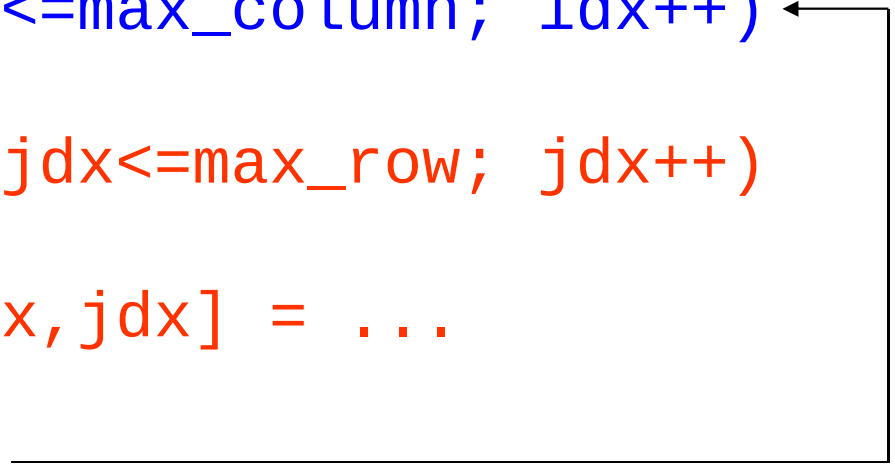
```
char  accept = [] ;

do{
/* loop body */
puts ( "Keep  going?  (y/n)  " ) ;

accept =  getchar ( ) ;

if( accept !=  'y' )
    break ;

}  while (/* other conditions */ ) ;
```

*Go out of the **whole** loop*

Break

exits **inner-most** loop

```
for (idx=1; idx<=max_column; idx++)
  {
   for (jdx=1; jdx<=max_row; jdx++)
     {
       matix[idx,jdx] = ...

       break; }
   ...
                                    }
```



continue

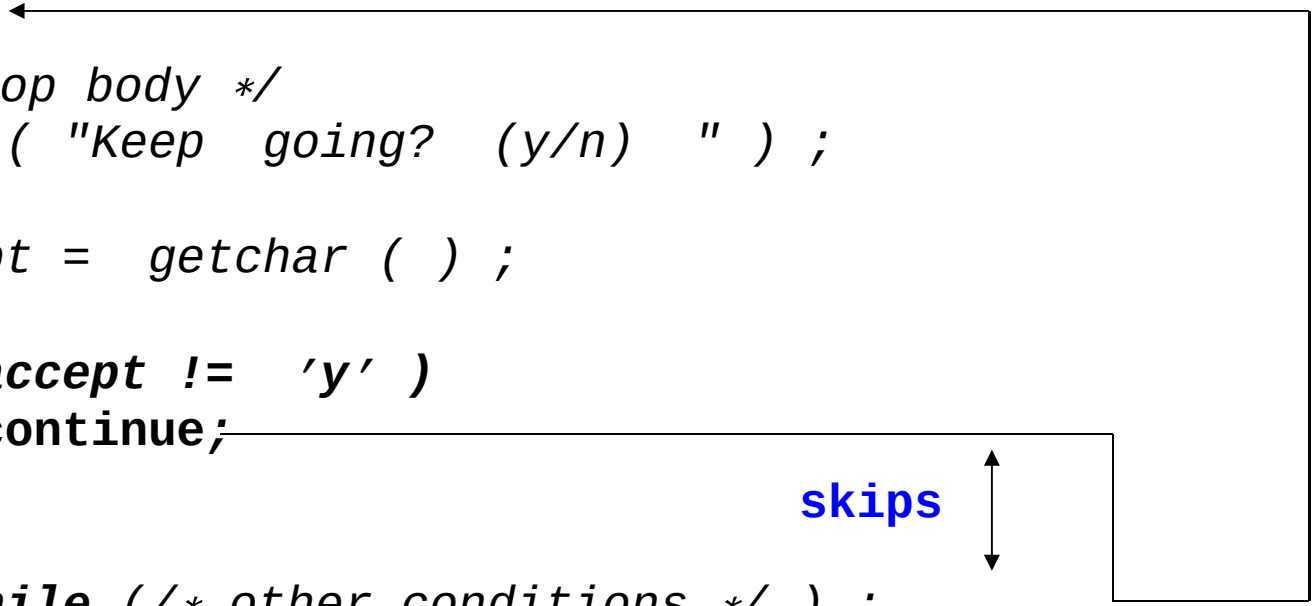
Use to skip an iteration

continue; skips **rest** of inner-most loop body, jumping to loop condition

```
char  accept = [] ;
```

*Go out of **this** loop,*
go on with the next loop

```
do{
/* loop body */
puts ( "Keep  going?  (y/n)  " ) ;

accept =  getchar ( ) ;

if( accept !=  'y' )
    continue;
…

}  while (/* other conditions */ ) ;
```



continue

exits **inner-most** loop

```
for (idx=1; idx<=max_column; idx++)
  {
   for (jdx=1; jdx<=max_row; jdx++)
     {
       matrix[idx,jdx] = ...

       continue;

       matrix  = …
     }
  ...
}
```

skips

Contents

Statements and Blocks
If-Else
Else-If
Switch
Loops - While and For
Loops - Do-While
Break and Continue
à Goto and labels

goto

C provides the infinitely-abusable goto statement, and labels to branch to. Formally, the goto statement is **never** necessary, and in practice it is almost always easy to write code **without** it.

The most common is to **abandon** processing in some deeply nested structure, such as breaking out of **two or more** loops at once. The **break** statement **cannot** be used directly since it only exits from the innermost loop.

A label has the same form as a **variable** name, and is followed by a **colon(:)**. It can be attached to any statement in the same function as the goto. The scope of a label is the **entire** function.

```
for ( ... )
        for ( ... ) {
            ...
            if (disaster)
                goto error;
        }
    ...
error:
    /* clean up the mess */
```

goto

As another example, consider the problem of determining whether two arrays **astring** and **bstring** have an element in common. One possibility is,

```
for (ind = 0; ind < n; ind ++)
          for (jnd = 0; jnd < m; jnd ++)
              if (astring[ind] == bstring[jnd])
                  goto found;
      /* didn't find any common element */
      ...


  found:
      /* got one: astring [ind] == bstring[jnd] */
      ...
```

goto

Code involving a **goto** can always be **re-written without** one, though perhaps at the price of some repeated tests or an extra variable. For example, the array search becomes,

```
int found = 0; /* 0 – NOT found; 1 – found */

for (ind = 0; ind < n; ind ++)
          for (jnd = 0; jnd < m; jnd ++)
              if (astring[ind] == bstring[jnd])
                  found = 1;
      /* didn't find any common element */
      ...

if (found)
      /* got one: astring[ind] == bstring[jnd]) */
      ...
  else
      /* didn't find any common element */
      ...
```

Exercise – 3.1

The real roots of the quadratic equation ***ax² + bx + c = 0*** are given by

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

only if ***b² > 4ac***. Write a program to calculate the real roots of a quadratic given the values of ***a***, ***b*** and ***c***. Your program should also ensure that no attempt to divide by zero is made.

Exercise – 3.2

Write a program to calculate how much to tip a waiter/waitress based on quality of service.

The c file should ask for the amount of the bill and whether the service was good (1), fair (2), or poor (3).

If the service was good, the tip should be 15% with a minimum tip of £2.

If the service was fair, the tip should be 10% with a minimum of £1.

If the tip was poor, the tip should be zero.

Exercise – 3.3

The body-mass index (BMI) was invented around 1840 by Adolphe Quetelet. It is meant to be used as a simple means of classifying sedentary individuals with an average body composition.

BMI is defined as :

Category	**BMI range - kg/m 2**
Severely underweight	*less than 16.5*
Underweight	*from 16.5 to 18.5*
Normal	*from 18.5 to 25*
Overweight	*from 25 to 30*
Obese Class I	*from 30 to 35*
Obese Class II	*from 35 to 40*
Obese Class III	*above 40*

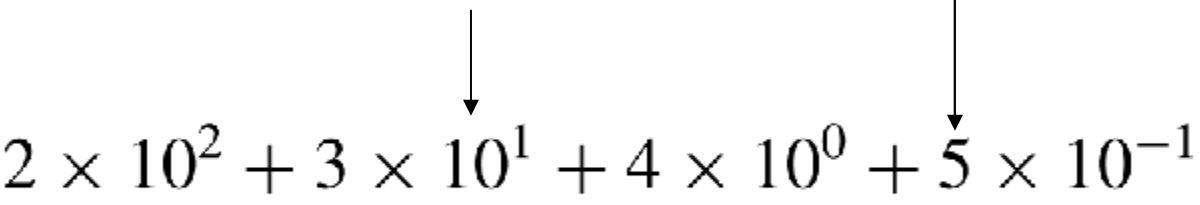
$$BMI = \frac{weight(kg)}{heigh^2(m^2)}$$

Write a program which asks for the user's height and weight, tells them which class they fall into and also tells them their optimum weight range (i.e. for a "normal" BMI).

FAQ 3-1

The decimal number 234.5 is equivalent to

the sum of terms comprising: (a **digit**)
multiplied by (the **base** raised to some power).



$$2 \times 10^2 + 3 \times 10^1 + 4 \times 10^0 + 5 \times 10^{-1}$$

FAQ 3-1

Conversion of **binary** to **decimal**

In the binary system of numbers, the **base** is **2**, so **1101.1**$_2$ is equivalent to:

$$1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1}$$

Thus the decimal number equivalent to the binary number **1101.1**$_2$ is

$$8 + 4 + 0 + 1 + \frac{1}{2}, \text{ that is } 13.5$$

FAQ 3-1

$$
\begin{aligned}
11011_2 &= 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 \\
&\quad + 1 \times 2^1 + 1 \times 2^0 \\
&= 16 + 8 + 0 + 2 + 1 \\
&= \mathbf{27_{10}}
\end{aligned}
$$

FAQ 3-1

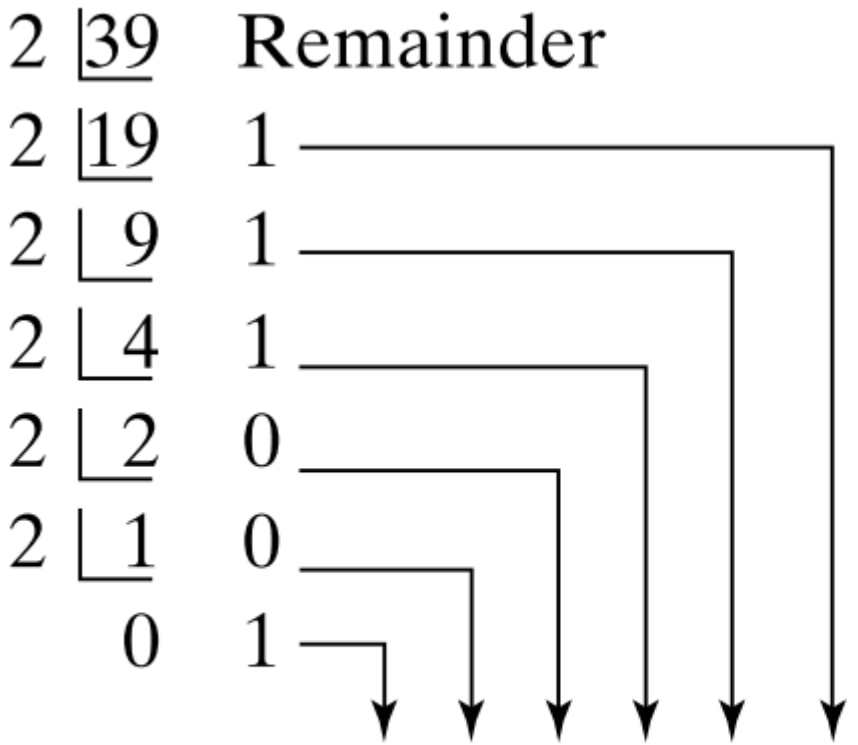
$$
\begin{aligned}
0.1011_2 &= 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} \\
&\quad + 1 \times 2^{-4} \\
&= 1 \times \frac{1}{2} + 0 \times \frac{1}{2^2} + 1 \times \frac{1}{2^3} \\
&\quad + 1 \times \frac{1}{2^4} \\
&= \frac{1}{2} + \frac{1}{8} + \frac{1}{16} \\
&= 0.5 + 0.125 + 0.0625 \\
&= \mathbf{0.6875_{10}}
\end{aligned}
$$

FAQ 3-1

$$
\begin{aligned}
101.0101_2 &= 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 \\
&\quad + 0 \times 2^{-1} + 1 \times 2^{-2} \\
&\quad + 0 \times 2^{-3} + 1 \times 2^{-4} \\
&= 4 + 0 + 1 + 0 + 0.25 \\
&\quad + 0 + 0.0625 \\
&= \mathbf{5.3125_{10}}
\end{aligned}
$$

FAQ 3-2

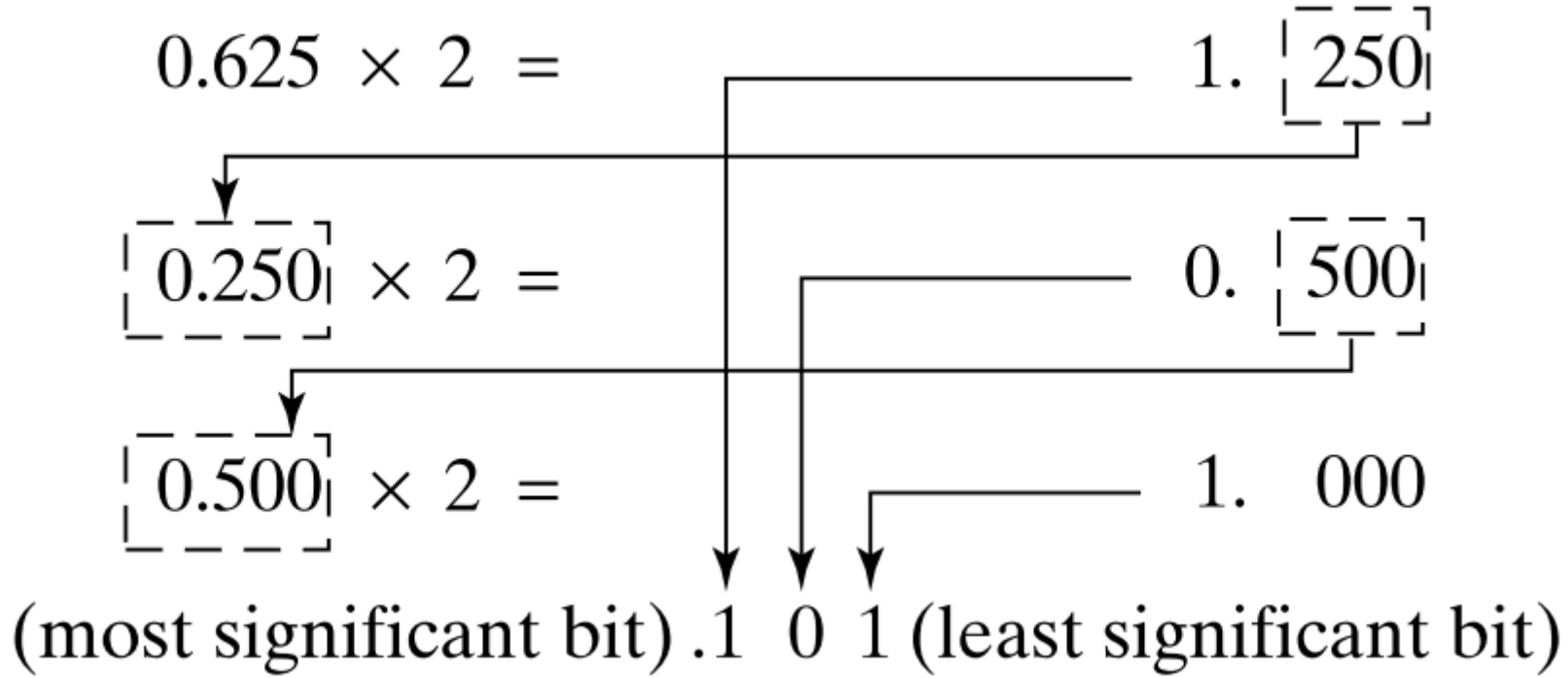
Conversion of **decimal** to **binary**



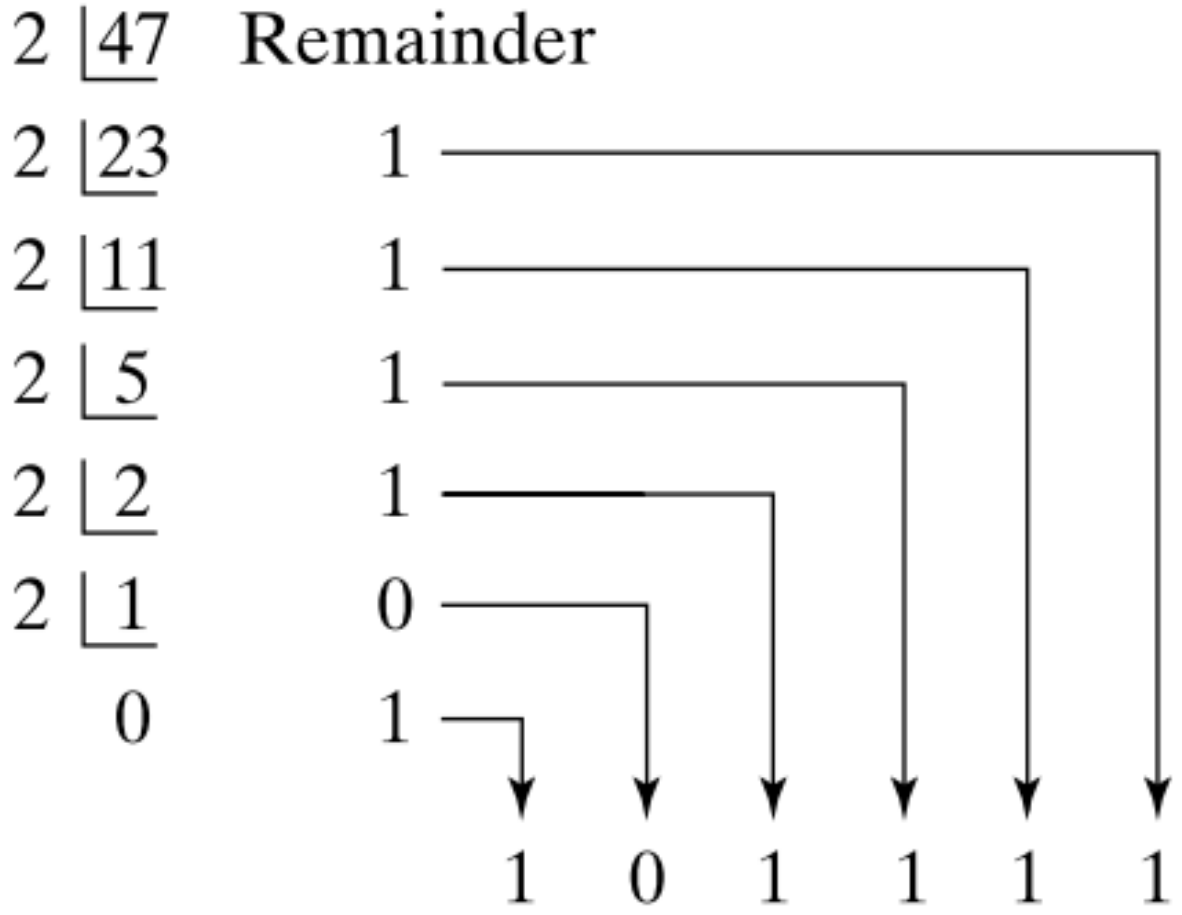
Thus $39_{10} = 100111_2$

(most → 1 0 0 1 1 1 ← (least
significant bit) significant bit)

FAQ 3-2

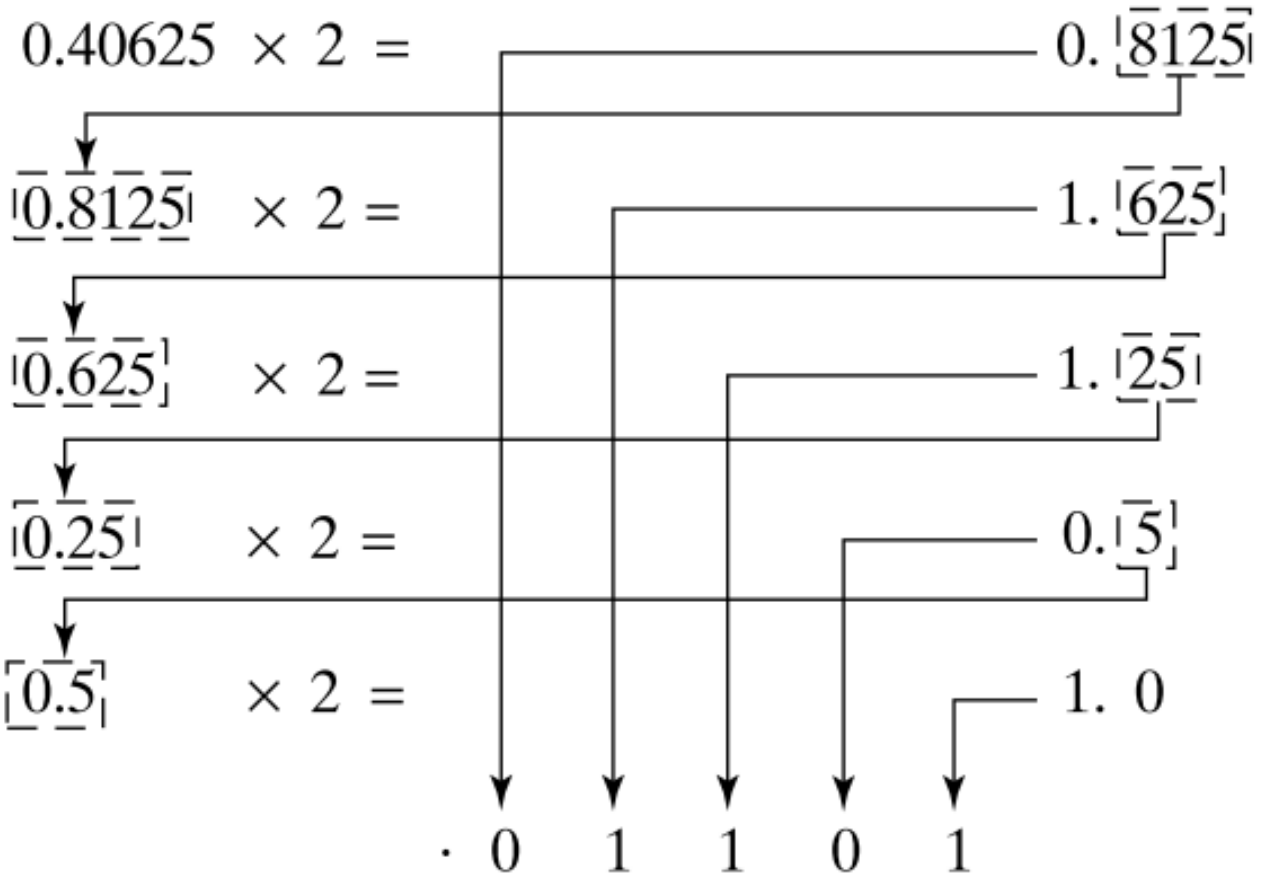

0.625 × 2 = 1. 250
0.250 × 2 = 0. 500
0.500 × 2 = 1. 000
(most significant bit) .1 0 1 (least significant bit)

FAQ 3-2



Thus $47_{10} = 101111_2$

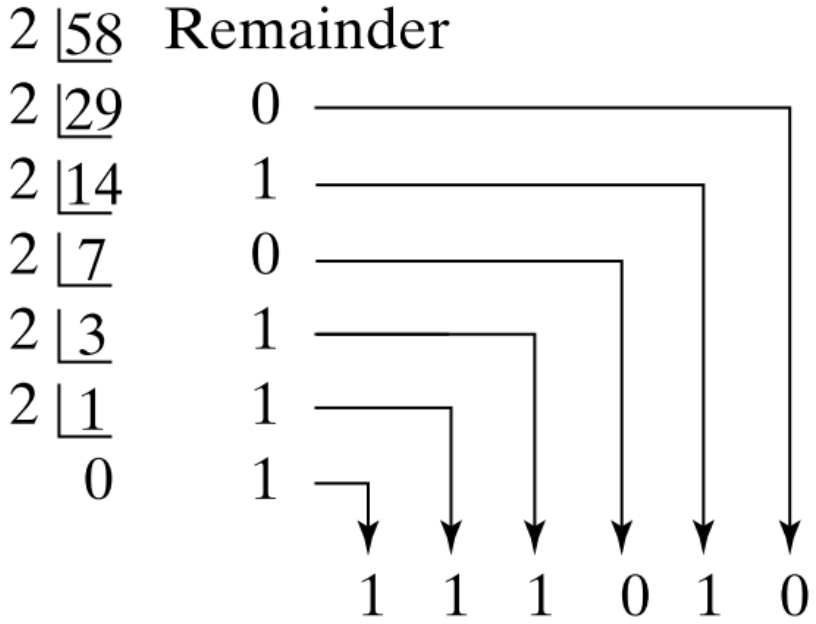
FAQ 3-2



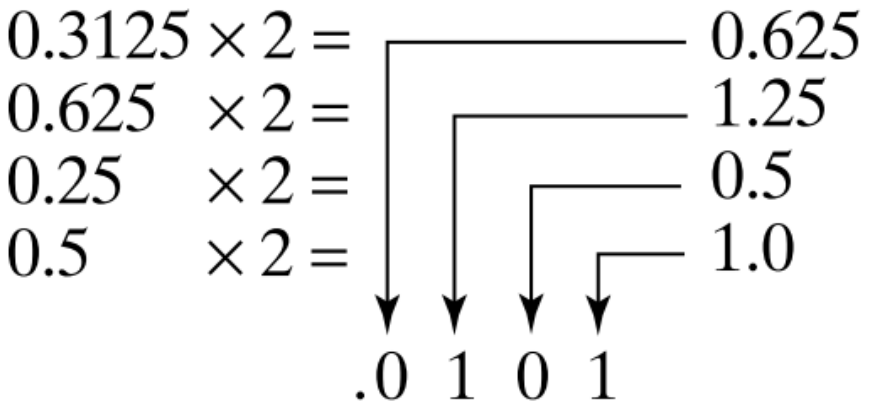
$$0.40625_{10} = 0.01101_2$$

FAQ 3-2

The **integer part** is repeatedly divided by 2



$$58.3125_{10} = 111010.0101_2$$



The **fractional part** is repeatedly multiplied by 2

FAQ 3-3

$4317_8 = 2255_{10}$

$$4 \times 8^3 + 3 \times 8^2 + 1 \times 8^1 + 7 \times 8^0$$

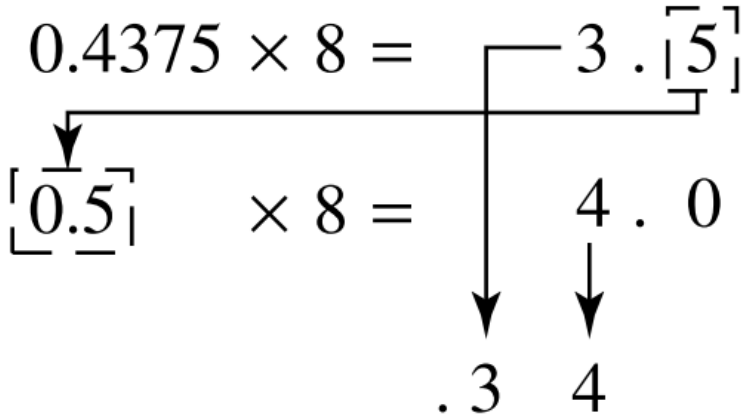
$$4 \times 512 + 3 \times 64 + 1 \times 8 + 7 \times 1 \text{ or } 2255_{10}$$

		Remainder
8 | 493		
8 | 61		5
8 | 7		5
0		7

7 5 5

$493_{10} = 755_8$

FAQ 3-3

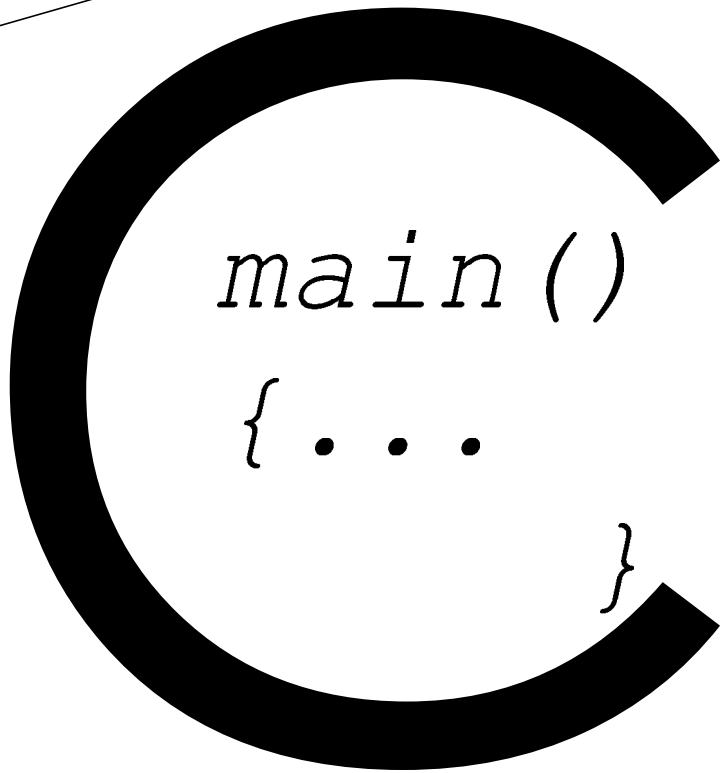


$0.4375_{10} = 0.34_8$

Decimal	Binary	Octal	Hexadecimal
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F
16	10000	20	10
17	10001	21	11
18	10010	22	12
19	10011	23	13
20	10100	24	14
21	10101	25	15
22	10110	26	16
23	10111	27	17
24	11000	30	18
25	11001	31	19
26	11010	32	1A
27	11011	33	1B
28	11100	34	1C
29	11101	35	1D
30	11110	36	1E
31	11111	37	1F
32	100000	40	20

C Programming Practice No[3]

Chen , Yi
leo.chen.yi@live.co.uk
30-Jul-2010



Control Flow

End

BETA 1.0.0.1


[木兰花令·拟古决绝词.One to Ten]#
一抹烟月孤灯上，
二三点雨秋湖浪。
四更夜静芳草香，
五六野雁栖苇荡。
七星北指晚寒畅，
八行长短寄盼望。
九照花镜理红妆，
十里相思吟吟唱。
2014-Feb-16