Chapter 2 Control Statements and Related Operators

Ground Rules

- Switch off your handphone and pager
- Switch off your laptop computer and keep it
- No talking while lecture is going on
- No gossiping while the lecture is going on
- Raise your hand if you have question to ask
- Be on time for lecture
- Be on time to come back from the recess break to continue the lecture
- Bring your lecturenotes to lecture

2.1 Logical Values and Relational Operators

- Control statements are used for
  - iteration (executing statements repeatedly)
  - selection (choosing which statements will be executed)
- Relational operators are used for testing the state of the computation. E.g.

```
if (a>2) printf ("a is greater than 2");
```

- logical operators are used for formulating more complicated tests. E.g., &&

```
if ((a>2) && (a<20))
   printf ("a is in the specified range");
```

```
if (a>2) printf ("a is greater than 2");
```

```
if (a>2)
    printf ("a is greater than 2");
else
    printf ("a is not greater than 2");
```

```
if (condition)
  {
          :
          :
  }
else
 {
          :
          :
  }
```

Relational Operators

Operator	Condition
==	is equal to
!=	is not equal to
<	is less than
<=	is less than or equal to
>	is greater than
>=	is greater than or equal to

$$\left.\begin{array}{c}\geq\\ \leq\\ \neq\end{array}\right\}\text{ All fatal!}$$

2.2 Iteration

while, do, for.

2.3 The while Statement

The while statement has the following form:

```
while (expression)
    statement;
```

The control expression is evaluated before each execution of the controlled statement.

For example, the statements

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

produce the following printout:

```
1  2  3  4  5  6  7  8  9  10
```

What if n is initialized to 11? Why ?

What will be the printed on the screen?

```
int i=2;

while (i<3) printf ("apple ");
```

```
int i;

while (i<3) printf ("apple ");
```

```
int i=-1;

while (i<3) {
   printf ("apple ");
   i=i+1;
}
```

```
int i=-1;

while (i++<3) printf ("apple ");
```

```
int i=-1;

while (++i<3) printf ("apple ");
```

```
int i=-1;

while (++i<3); printf ("apple ");
```

2.4 The do Statement

The do statement has the following form (note the semicolon following the parenthesized control expression):

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

The do statement is similar to the while statement except that the control expression is evaluated after each execution of the controlled statement rather than before.

For example, the statements

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

If n is initialized to 11 in this example, what will be printed ?

produce the following printout:

```
1  2  3  4  5  6  7  8  9  10
```

What will be the printed on the screen?

```
int i=2;

do printf ("apple") while (i<3);
```

```
int i;

do printf ("apple") while (i<3);
```

```
int i=-1;

do{
    printf ("apple");
    i=i+1;
} while (i<3)
```

```
int i=-1;

do printf ("apple") while (i++<3) ;
```

```
int i=-1;

do printf ("apple") while (++i<3);
```

```
int i=-1;

do while (++i<3); printf ("apple");
```

2.5 The for Statement

- The for statement is intended for stepping a control variable through a series of values and executing a controlled statement once for each value. The for statement has the following form:

```
    for (exp-i; exp-t; exp-s)
      statement;
```

- Expression exp-i assigns the control variable its initial value, exp-t tests whether the iterations should continue, and exp-s steps the control variable from one value to the next.

  For example, in

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

  exp-i (n = 1) assigns n the initial value 1, exp-t (n <= 10) ensures that the value of n will not exceed 10, and exp-s (n = n + 1) increments the value of n after each execution of the controlled statement. The printout will be

```
   1  2  3  4  5  6  7  8  9  10.
```

- Each of the three expressions in a for statement can be omitted if it is not needed;
- If exp-t is omitted, the iterations will continue indefinitely.
- The most common omission is exp-i, which is not needed if the control variable has already been initialized by earlier statements.
- The two semicolons in the for statement are retained even when expressions are omitted, that is, for (; ;).

Steps of Execution in a for Loop

```
for (i=1;  i<=3; i++)
{
   printf ("\n loop  %d", i);
   printf ("\n ...... End of this loop");
}
```

i is assigned to 1
i<=3 is checked
if true execute loop body, else exit loop
i is incremented by 1
go to step 2

What is the screen output of the following code segment ?

```
for (i=3;  i<=2; i++)
{
   printf ("\n loop  %d", i);
   printf ("\n ...... End of this loop");
}
```

2.6 Increment, Decrement, and Compound Assignment Operators

p += n is equivalent to p = p + n

p -= n is equivalent to p = p - n

p *= n is equivalent to p = p * n

p /= n is equivalent to p = p / n

p %= n is equivalent to p = p % n

2.7The Increment and Decrement Operators

```
n = n + 1;
n += 1;
++n;
n++;
```

Likewise, the following expressions are equivalent:

```
n = n - 1;
n -= 1;
--n;
n--;
```

E.g.:

```
sum = 0;
for (n = 1; n <= 5; n++)
{
   scanf("%d", &number);
   sum += number;            /* sum = sum +
     number */
}
```

can be used to sum 5 numbers entered from keyboard.

Note: If we use the value returned by the increment or decrement operator, we must choose the prefix or postfix operator depending on the value we need.

For example, the statements

```
p = ++n;        means n = n+1; followed by p=n;
p = n++;        means p=n; followed by n = n+1;
```

are NOT equivalent. Both increment the value of n. But the first assigns to p the new incremented value of n, whereas the second assigns the old non-incremented value.

Consider the following statements executed in sequence:

```
int p1, p2, n=5;
p1 = ++n;
p2 = n++;
```

After the first assignment statement p1 = ++n, the values of p1 and n are both equal to 6. After the second assignment statement p2 = n++, p2 = 6 whereas n = 7.

How about n = n++ ?   // concept of side effect

You are advised to be moderate when you use the ++ and -- operators because the ease of read of a program is important.

2.8 Examples Using Iteration

Consider a fibonacci sequence, where the value of a number is the sum of the previous two numbers.

a1 = 1

a2 = 1

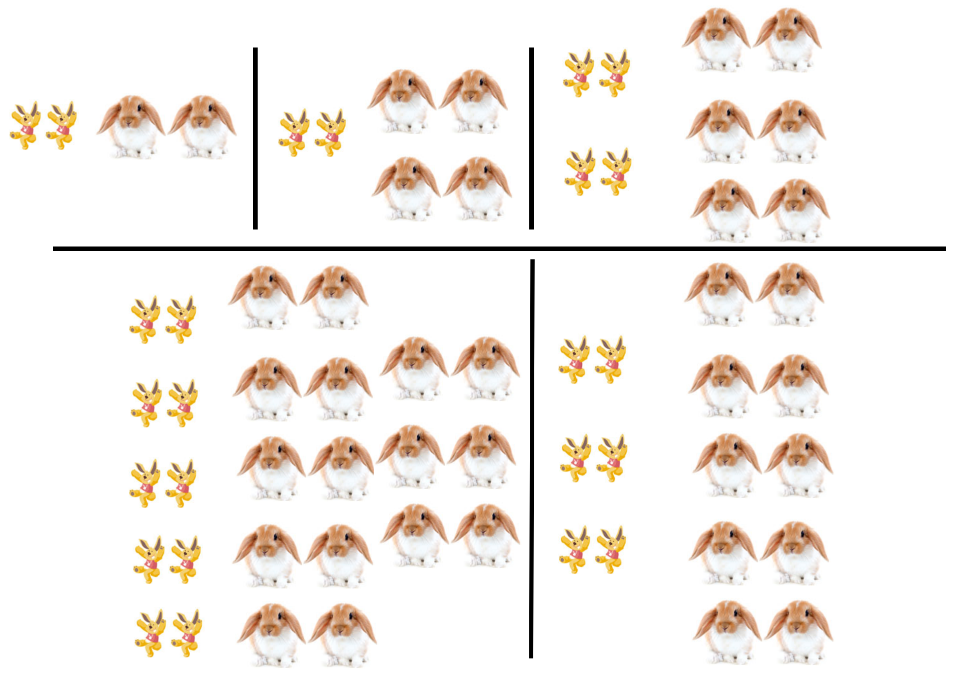
a3 =

a4 =

a5 =

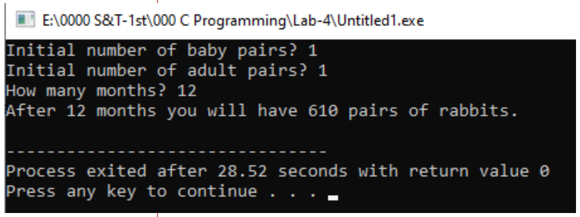
:

:



L2-1.c

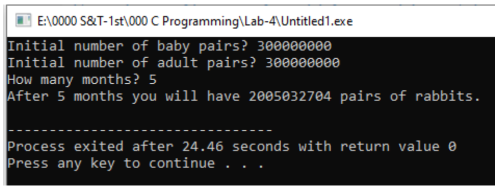
```
/* File months.c */
/* Solve Fibonacci rabbit problem */
#include <stdio.h>
int main(void)
{
   int num_of_babies, num_of_adults;
   int this_month_total;
   int months, m;
   int total;

    printf("Initial number of baby pairs? ");
   scanf("%d", &num_of_babies);
   printf("Initial number of adult pairs? ");
   scanf("%d", &num_of_adults);
   printf("How many months? ");
   scanf("%d", &months);

   for (m = 1; m <= months; m++)
   {  /* At the beginning of the month */
     this_month_total = num_of_babies +  num_of_adults;

      /* end of this month (or for the  beginning of next month) */
     num_of_babies = num_of_adults;
     num_of_adults = this_month_total;
   }
   total = num_of_babies + num_of_adults;
   printf("After %d months you will have %d pairs of rabbits.\n", months, total);
   return 0;
}
```



E:\0000 S&T-1st\000 C Programming\Lab-4\Untitled1.exe

```
Initial number of baby pairs? 1
Initial number of adult pairs? 1
How many months? 12
After 12 months you will have 610 pairs of rabbits.

--------------------------------
Process exited after 28.52 seconds with return value 0
Press any key to continue . . .
```

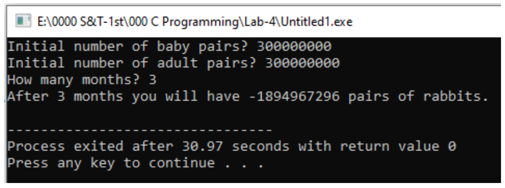
(300,000,000, 300,000,000, 5)



E:\0000 S&T-1st\000 C Programming\Lab-4\Untitled1.exe

```
Initial number of baby pairs? 300000000
Initial number of adult pairs? 300000000
How many months? 5
After 5 months you will have 2005032704 pairs of rabbits.

--------------------------------
Process exited after 24.46 seconds with return value 0
Press any key to continue . . .
```

(300,000,000, 300,000,000, 3)



E:\0000 S&T-1st\000 C Programming\Lab-4\Untitled1.exe

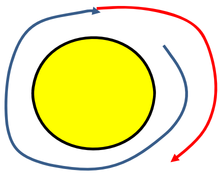
```
Initial number of baby pairs? 300000000
Initial number of adult pairs? 300000000
How many months? 3
After 3 months you will have -1894967296 pairs of rabbits.

--------------------------------
Process exited after 30.97 seconds with return value 0
Press any key to continue . . .
```

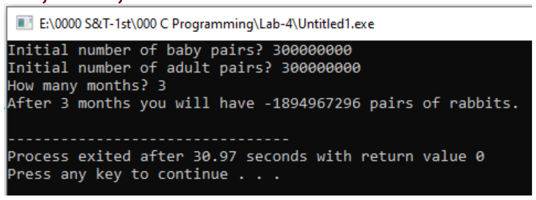
In DEV C++, integer has 32 bits.

Thus it has $2^{32} = 4,294,967,296$ representations.

The range is from -2,147,483,648 to 2,147,483,647.

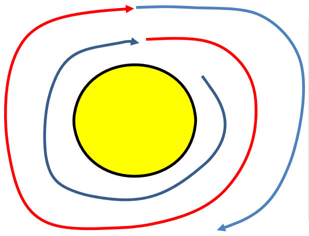
$$f_{-2} = 300,000,000$$
$$f_{-1} = 300,000,000$$
$$f_0 = 600,000,000$$
$$f_1 = 900,000,000$$
$$f_2 = 1,500,000,000$$
$$f_3 = 2,400,000,000$$
$$f_4 = 3,900,000,000$$
$$f_5 = 6,300,000,000$$

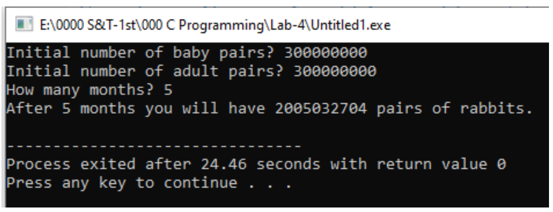




```
Initial number of baby pairs? 300000000
Initial number of adult pairs? 300000000
How many months? 3
After 3 months you will have -1894967296 pairs of rabbits.

--------------------------------
Process exited after 30.97 seconds with return value 0
Press any key to continue . . .
```

(-1,894,967,296), or (2400000000 - 4294967296)





```
Initial number of baby pairs? 300000000
Initial number of adult pairs? 300000000
How many months? 5
After 5 months you will have 2005032704 pairs of rabbits.

--------------------------------
Process exited after 24.46 seconds with return value 0
Press any key to continue . . .
```

(2,005,032,704), or (6300000000 - 4294967296)

$$f_4 = 3,900,000,000$$
$$f_5 = 6,300,000,000$$
$$f_6 = 10,200,000,000$$

But the display is 1,610,065,408. It is positive and you think the calculation is correct again. How is the calculation wrong?

Answer: 10,200,000,000 – (4,294,967,296 x 2) = 1,610,065,408.

$$f_7 = 16,500,000,000$$

But the display is – 679,869,184. How is the calculation wrong?

Answer: 16,500,000,000 – (4,294,967,296 x 4) = – 679,869,184.

$$\left.\begin{array}{l} f_8 = 26,700,000,000 \\ f_9 = 43,200,000,000 \\ \quad\quad\vdots \\ \quad\quad\vdots \end{array}\right\}$$

The wrong calculations can be explained by the cyclic property of the 2's complement notation.

Computer is a finite machine!!
- A/Prof Tay

计算机是一座有限的机器!!
- A/Prof Tay

L2-2.c

```
/* Solve inverse Fibonacci rabbit problem */
/* inverse.c */
 #include <stdio.h>
 int main(void)
{
   int num_of_babies, num_of_adults, this_month_total;
   int needed;
   int months = 0;
   printf("Initial number of baby pairs? ");
   scanf("%d", &num_of_babies);
   printf("Initial number of adult pairs? ");
   scanf("%d", &num_of_adults);
   printf("How many pairs do you need? ");
   scanf("%d", &needed);
    this_month_total = num_of_babies + num_of_adults;
    while (this_month_total < needed)
    {
       /* next month */
       num_of_babies = num_of_adults;
       num_of_adults = this_month_total;
       months++;
       this_month_total = num_of_babies + num_of_adults;
    }
   printf("After %d months you will have %d pairs of rabbits.\n",months, this_month_total);
   return 0;
}
```

Screen Output:

```
Initial number of baby pairs? 1
Initial number of adult pairs? 1
How many pairs do you need? 5
After 2 months you will have 5 pairs of rabbits.
```

You open an bank account with $100.

On the first day of every month, you deposit $10.

If the monthly interest rate is 10%, what is your balance at the end of the 4th month? Assume that monthly compound is used.

Balance = $100 + $10 = $110

Interest = $110 x 0.1 = $11

New Balance = $110 + $11 = $121

Balance = $121 + $10 = $131

Interest = $131 x 0.1 = $13.1

New Balance = $131 + $13.1 = $144.1

Balance = $144.1 + $10 = $154.1

Interest = $154.1 x 0.1 = $15.41

New Balance = $154.1 + $15.41 = $169.51

Balance = $169.51 + $10 = $179.51

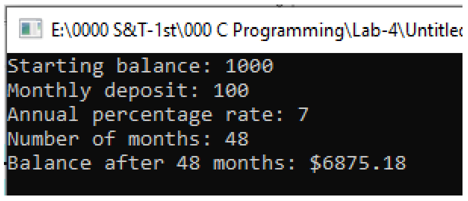
Interest = $179.51 x 0.1 = $17.951

New Balance = $179.51 + $17.951 = $197.461

What will be your balance at the end of 20 months?

L2-3.c

```
 /* File balance.c */
/* Computer amount in bank account */
 #include <stdio.h>
 #define APR_TO_MDR   1200.0    /* For converting
  annual percentage rate to monthly decimal rate */
 int main(void)
{
   float balance;              /* Current balance */
   float deposit;              /* Monthly deposit */
   float annual_pcnt_rate;   /* Annual percentage rate */
   float rate;                 /* Monthly decimal rate */
   float interest;             /* Interest for this month */
   int months;                 /* Number of months */
   int m;                      /* Current month */
    /* Get input data from user */
    printf("Starting balance: ");
   scanf("%f", &balance);
   printf("Monthly deposit: ");
   scanf("%f", &deposit);
   printf("Annual percentage rate: ");
   scanf("%f", &annual_pcnt_rate);
   printf("Number of months: ");
   scanf("%d", &months);

 /* Compute new balance */
    rate = annual_pcnt_rate / APR_TO_MDR;
    for(m = 1; m <= months; m++)
    {
      balance += deposit;
      interest = balance * rate;
      balance += interest;
    }
    printf("Balance after %d months: $%.2f\n",
              months, balance);
   return 0;
 }
```



```
Starting balance: 1000
Monthly deposit: 100
Annual percentage rate: 7
Number of months: 48
Balance after 48 months: $6875.18
```

2.9 Selection Statements and the Conditional Operator

if, if else, switch.

2.10 The if and if else Statements

The if statement has the form

```
if (expression)
    statement;
```

The control expression is evaluated first. If the value of the control expression is *true*, the controlled statement is executed; if the value is *false*, the controlled statement is ignored.

Adding an else part to the if statement gives us the if-else statement:

```
if (expression)
   statement-l;
else
   statement-2;
```

If the value of the control expression is true, statement-l is executed; if the value of the expression is false, statement-2 is executed.

For example,

```
if (count < MAX_COUNT)
  count++;
else
  {
    printf("Overflow\n");
    count = 0;
 }
```

2.11 Nested if and if-else Statements

Nested if-else statements allow us to construct multiway selections, such as the following:

```
if (expression-l)
   statement-l;
else if (expression-2)
   statement-2;
else if  (expression-3)
   statement-3;
      .
      .
      .
else
   statement-n;
```

L2-5.c

```c
/* File areas1.c */
#include <stdio.h>

void print_menu(void);
#define TRUE          1   /* Logical values */
#define FALSE         0
#define RECTANGLE   1   /* Command numbers */
#define TRIANGLE      2
#define CIRCLE         3
#define QUIT           4
#define PI                          3.14159265
int main(void)
{
  int command;
  int running = TRUE;   /* flag */
  float length, width, base, height, radius, area;
```

```c
do
  {
    print_menu();
    scanf("%d", &command);
    if (command == RECTANGLE)
     {
      printf("Length and width? ");
      scanf("%f%f", &length, &width);
      area = length * width;
      printf("Area is %.1f\n", area);
     }
   else if (command == TRIANGLE)
    {
       printf("Base and height? ");
       scanf("%f%f", &base, &height);
       area = 0.5 * base * height;
       printf("Area is %.1f\n", area);
    }
   else if (command == CIRCLE)
    {
      printf("Radius? ");
      scanf("%f", &radius);
      area = PI * radius * radius;
      printf("Area is %.1f\n", area);
    }
   else if (command == QUIT)
    {
      running = FALSE;
      printf("Have a nice day!\n");
     }
    else
       printf("Invalid command--please try again\n");
  } while (running);
  return 0;
}
```

```c
void print_menu(void)
{
  printf("\n\n\n");
  printf("1.   Compute Area of Rectangle\n");
  printf("2.   Compute Area of Triangle\n");
  printf("3.   Compute Area of Circle\n");
  printf("4.   Quit\n\n");
  printf("Enter number of command: ");
}
```

Screen Output:

```
1.  Compute Area of Rectangle
2.  Compute Area of Triangle
3.  Compute Area of Circle
4.  Quit

Enter number of command: 1
Length and width? 75 20
Area is 1500.0

1.  Compute Area of Rectangle
2.  Compute Area of Triangle
3.  Compute Area of Circle
4.  Quit

Enter number of command: 100
Invalid command--please try again

1.  Compute Area of Rectangle
2.  Compute Area of Triangle
3.  Compute Area of Circle
4.  Quit

Enter number of command: 4
Have a nice day!
```

2.12 The switch and break Statements

The switch statement has the following general form:

```
switch (expression)
   statement
```

The controlled statement is normally a block, one or more of whose statements are preceded by case labels. In the following examples, each ellipsis (...) represents any number of statements.

Example 1:

```
option = 2;
switch (option)
{
  case 1:
    . . .
  case 2:
    . . .
  case 3:
    . . .
}
```

In this example, execution of the block begins immediately after the label case 2:

The case label default: corresponds to any value of the control expression that is not represented by another case label.

Example 2:

```
switch (option)
{
  case 1:
    . . .
  case 2:
    . . .
  default:
    . . .
}
```

If option = 3 in example 2, execution will begin following default: because none of the other case labels corresponds to the value (2) of the control expression.

The break statement directs the computer to exit from a switch statement or an iteration statement and continue with the rest of the program. The statements for each case should end with a break statement:

Example 3:

```
option = 2;

switch (option)
{
  case 1:
    . . .
    break;
  case 2:
    . . .
    break;
  case 3:
    . . .
    break;
}
```

In this example, only the statements for case2: are executed. In example 1, without the break statements, the statements for both case2: and case3: are executed.

2.13 Responding to Menu Selections

This program displays a menu. The user then enters the selection, as prompted. If the user enters an invalid command number, the program should print an error message. Normally the menu is displayed iteratively until a quit option is selected.

L2-6.c

```
/* File areas2.c */
#include <stdio.h>
void print_menu(void);
#define TRUE                1  /* Logical values */
#define FALSE               0
#define RECTANGLE   1  /* Command numbers */
#define TRIANGLE    2
#define CIRCLE      3
#define QUIT        4
 #define PI 3.14159265
 int main(void)
{
  int command;
  int running = TRUE;  /* flag */
  float length, width, base, height, radius, area;
do
{
    print_menu();
    scanf("%d", &command);
    switch (command)
    {
       case RECTANGLE:
          printf("Length and width? ");
          scanf("%f%f", &length, &width);
          area = length * width;
          printf("Area is %.1f\n", area);
          break;
       case TRIANGLE:
          printf("Base and height? ");
          scanf("%f%f", &base, &height);
          area = 0.5 * base * height;
          printf("Area is %.1f\n", area);
          break;
      case CIRCLE:
          printf("Radius? ");
          scanf("%f", &radius);
          area = PI * radius * radius;
          printf("Area is %.1f\n", area);
          break;
      case QUIT:
          running = FALSE;
          printf("Have a nice day!\n");
          break;
      default:
          printf("Invalid command--please try again\n");
          break;       /* optional  */
    } /* switch */
 } while (running);
 return 0;
}
```

```
void print_menu(void)
{
  printf("\n\n\n");
  printf("1.   Compute Area of Rectangle\n");
  printf("2.   Compute Area of Triangle\n");
  printf("3.   Compute Area of Circle\n");
  printf("4.   Quit\n\n");
  printf("Enter number of command: ");
}
```

Screen Output:

```
1.  Compute Area of Rectangle
2.  Compute Area of Triangle
3.  Compute Area of Circle
4.  Quit

Enter number of command: 1
Length and width? 75 20
Area is 1500.0

1.  Compute Area of Rectangle
2.  Compute Area of Triangle
3.  Compute Area of Circle
4.  Quit

Enter number of command: 100
Invalid command--please try again

1.  Compute Area of Rectangle
2.  Compute Area of Triangle
3.  Compute Area of Circle
4.  Quit

Enter number of command: 4
Have a nice day!
```

2.14 Logical Operators

- The logical operators are && (logical AND), || (logical OR), and ! (logical NOT).
- Like the relational operators, the logical operators help us construct expressions representing conditions.
- The logical AND, OR, and NOT operators must be carefully distinguished from the bitwise AND, OR, and NOT operators.

The AND (&&) Table

&&	True	False
True	True	False
False	False	False

Both must be true to get true

The OR (||) Table

||	True	False
True	True	True
False	True	False

Either or both true to get true

The NOT (!) Table

	True	False
!	False	True

- The bitwise operators are not used for representing conditions but for manipulating the individual bits that represent an integer value. This will be taught in part 2.
- The logical AND operator, &&, returns *true* only if both of its operands are *true*. If either operand is *false*, the operator returns *false*.
- The logical OR operator, ||, returns *true* if either of its operands is *true* or if both of them are *true*. It returns *false* only if both its operand are *false*.
- The logical NOT operator, !, returns *true* when its operand is *false* and vice versa.

For examples, if m = 3, n = 5, then

(i) (m > n) && (m < 0) returns false

(ii) (m == 3) || (n != 5) returns true

(iii) !(m > 0 && n > 0) returns false

2.15 Short-Circuit Evaluation

The operators && and || are additional examples of operators whose operands are evaluated in a particular order. It may not be necessary to evaluate the right operand at all. If the left operand of && is *false*, the operator returns *false*; and if the left operand of || is *true*, the operator returns *true*. In these cases the right operand is not evaluated. This process is known as short-circuit evaluation.

Example:

```
a = 0;

if ( (a >0 && b > 6) && c !=78 )
   d = b*c;
else
   d=2*b;
```

L2-7.c

```
 /* File triang_2.c */
/* Classify triangles */
 #include <stdio.h>

 int classify(float, float, float);

/* Codes for classifications */

#define EQUILATERAL     1
#define ISOSCELES        2
#define SCALENE          3
#define END_OF_DATA   0.0

int main(void)
{
  float s1, s2, s3;
  printf("Enter three sides (or 0 to stop): ");
  scanf("%f", &s1);
```

```
   while (s1 != END_OF_DATA)
   {
       scanf("%f%f", &s2, &s3);
       if (s1+s2 >s3 && s2+s3 >s1 && s3+s1 >s2)
         switch (classify(s1, s2, s3))
         {
           case EQUILATERAL:
                   printf("Equilateral\n");
                   break;
            case ISOSCELES:
                   printf("Isosceles\n");
                   break;
            case SCALENE:
                   printf("Scalene\n");
                break;
         }  /* switch */
      else
        printf ("Not a triangle\n");

        printf("Enter three sides (or 0 to stop): ");
        scanf("%f", &s1);
   } /* while */
   return 0;
 }
```

```
/* Return integer code for kind of triangle */

int classify(float s1, float s2, float s3)
{
  if ((s1 == s2) && ( s2 == s3))
    return EQUILATERAL;
  else if ((s1 == s2) || (s2 == s3) || (s1 == s3))
    return ISOSCELES;
  else
    return SCALENE;
}
```

Screen Output:

```
Enter three sides (or 0 to stop): 1.0 2.0 3.0
Not a triangle
Enter three sides (or 0 to stop): 2.0 2.0 3.0
Isosceles
Enter three sides (or 0 to stop): 3.0 3.0 3.0
Equilateral
Enter three sides (or 0 to stop): 0
```

2.14 The Conditional Operator

The conditional operator has three operands, which are separated by the symbols ? and : as follows:

```
exp-c ? exp-t : exp-f
```

Exp-c is the control expression; its value determines which of the two following expressions will be evaluated. If exp-c yields *true*, exp-t is evaluated and its value is returned by the conditional operator. If exp-c yields *false*, exp-f is evaluated and its value is returned.

For example, the following expression yields the maximum of the values of m and n:

```
m >= n ? m : n
```

Likewise, the expression

```
m > 0 ? m : -m
```

returns the absolute value of m.

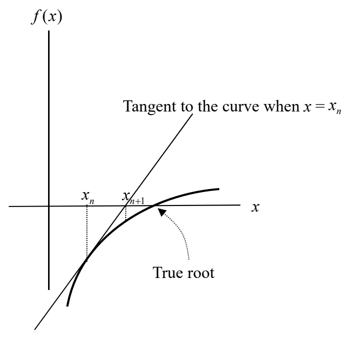
Eg: y = (m > 0) ? m : -m

This is equivalent to:

```
if (m > 0)
   y = m;
else
   y = -m;
```

Numerical Method for finding the Root of Equation

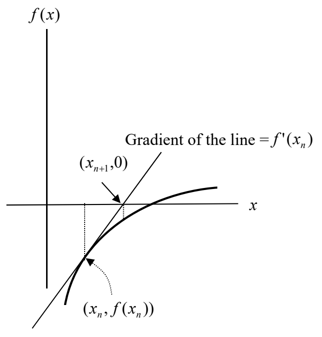
To find the root of the equation $f(x)$ shown in the diagram on the right, the Newton–Raphson's method starts with an initial guess which is close to the true root, and compute the subsequent points to come closer to the solution. This is done by drawing a tangential line on the initial point and determining the x-intercept of the tangent. The x-intercept is the second point and can be a better approximation of the root than the first guess, and the method is repeated until a close tolerance is reached.

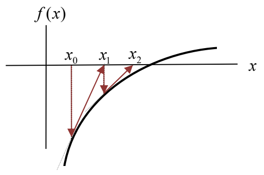


Numerical Method for finding the Root of Equation

In general, the next point of the approach towards the true root is expressed as follows:

$$f'(x_n) = \frac{0 - f(x_n)}{x_{n+1} - x_n} \rightarrow x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$





In the program named as **newton.c**, it uses the Newton-Raphson's method to find the root of $f(x) = e^{2x} - x - 6.$

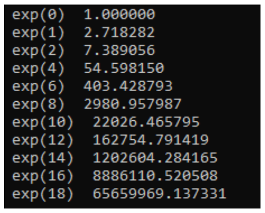
The initial guess is $x_0 = 0.25.$

The iteration can stop when $|f(x_n)| < 0.000001.$

Please take note that the Newton-Raphson's method may not work if the initial point does not lead to the convergence of the solution or the gradient of the tangential lines fluctuates in unhelpful patterns.

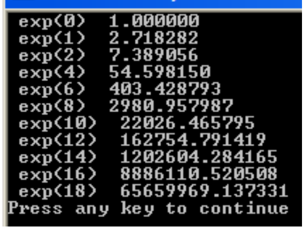
An example on the use of exponential function is as follows:

```
// exponent.c
#include<stdio.h>
#include<math.h>
int main()
{
   double x=1;
   int i;
   printf (" exp(%d)  %lf \n", 0, exp(0));
   for (i=1; i<=10; x=i*2, i++) printf (" exp(%d)  %lf \n", (int) x, exp(x) );
   return 0;
}
```



E:\0000 S&T-1st\000 C Programming\

```
exp(0)  1.000000
exp(1)  2.718282
exp(2)  7.389056
exp(4)  54.598150
exp(6)  403.428793
exp(8)  2980.957987
exp(10)  22026.465795
exp(12)  162754.791419
exp(14)  1202604.284165
exp(16)  8886110.520508
exp(18)  65659969.137331
```

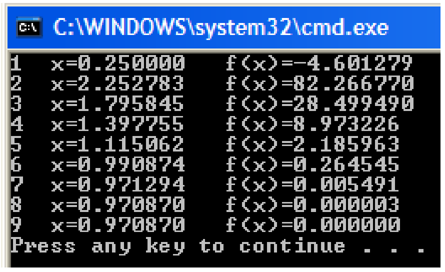


C:\WINDOWS\system32\cmd.e

```
 exp(0)  1.000000
 exp(1)  2.718282
 exp(2)  7.389056
 exp(4)  54.598150
 exp(6)  403.428793
 exp(8)  2980.957987
 exp(10)  22026.465795
 exp(12)  162754.791419
 exp(14)  1202604.284165
 exp(16)  8886110.520508
 exp(18)  65659969.137331
Press any key to continue
```

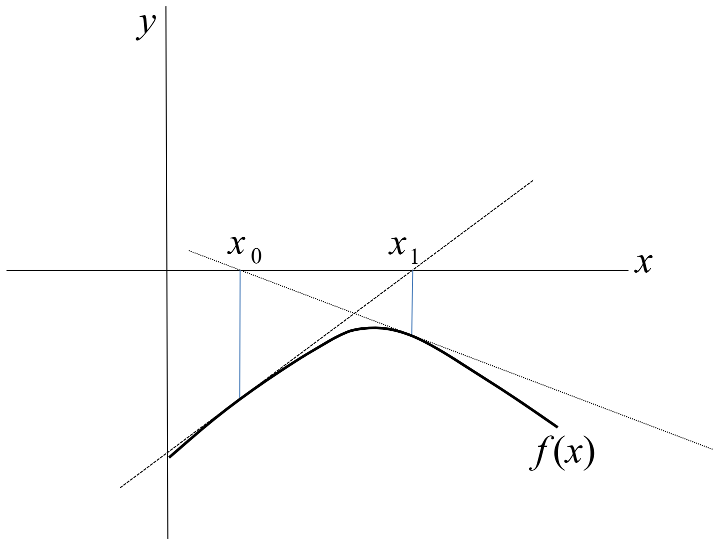
```
#include<stdio.h>
#include<math.h>
#define f(x) (exp(2*(x)) - (x) -6)
#define fpr(x) (2*exp(2*(x)) -1)
#define last_round 100
#define start 0.25
#define tolerance 0.000001
int main()
{
  double x1, x2;
  int rounds;
  rounds=1;
  x1=start;
  while ( (f(x1) > tolerance || f(x1) < -tolerance) && (rounds<=last_round))
  {
    printf ("%d  x=%lf   f(x)=%lf\n", rounds, x1, f(x1));
    x2 = x1 - f(x1)/fpr (x1);
    x1 = x2;
    rounds++;
  }
  printf ("%d  x=%lf   f(x)=%lf\n", rounds, x1, f(x1));
  return 0;
}
```

$$f(x) = e^{2x} - x - 6.$$


C:\WINDOWS\system32\cmd.exe

```
1  x=0.250000   f(x)=-4.601279
2  x=2.252783   f(x)=82.266770
3  x=1.795845   f(x)=28.499490
4  x=1.397755   f(x)=8.973226
5  x=1.115062   f(x)=2.185963
6  x=0.990874   f(x)=0.264545
7  x=0.971294   f(x)=0.005491
8  x=0.970870   f(x)=0.000003
9  x=0.970870   f(x)=0.000000
Press any key to continue . . .
```

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$



The iterations may not converge to a solution and can be trapped in an infinite loop.

Random Number（随机数字）

- Suppose the time required to serve a customer is 2 to 3 minutes, with an uniform distribution of mean 2.5 minutes. What is the service times required by the first 4 customers?

Concept of Random Number and Simulation (cont'd)

Answer:

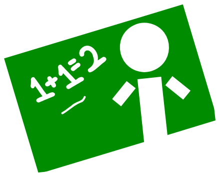
These are random numbers from 2 to 3. We can only ensure that the service times are uniformly distributed, and in the long run the mean is equal to 2.5 (minutes).

(Pseudo冒充) Random Number Generator

Mid-Square Method

An initial number (seed) is squared, and the middle digits of this square become the random number after the placement of the decimal. The middle digits are then squared to generate the second random number. The technique continues in the same fashion.

Example:



$X_0 = 5497$

$X_0^2 = 5497^2 = 30217009 \rightarrow X_1=2170$

$R_1 = 0.2170$

$X_1^2 = 2170^2 = 04708900 \rightarrow X_2=7089$

$R_2 = 0.7089$

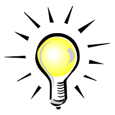
$X_2^2 = 7089^2 = 50253921 \rightarrow X_3 = 2539$

$R_3 = 0.2539$

Drawbacks of Mid-Square Method

1. Biased Distribution

$X_0 = 5197$
$X_0^2 = 5197^2 = 27008809 \rightarrow X_1=0088$
$R_1= 0.0088$



$X_1^2 = 88^2 = 00007744 \rightarrow X_2=0077$
$R_2= 0.0077$

$X_2^2= 77^2=00005929 \rightarrow X_3 = 0059$
$R_3 = 0.0059$

The leading zeros will appear in every succeeding R_i.

2. Degeneration (退化)

$X_i = 6500$
$X_i^2 = 6500^2 = 42250000 \rightarrow X_{i+1}=2500$
$R_i= 0.2500$

$X_{i+1}^2 = 2500^2 = 06250000 \rightarrow X_{i+2}=2500$
$R_{i+1}=0.2500$

This is a degenerating (退化) condition since all subsequent values of X_i will be 2500.

Other Random Number Generation Methods

- Mid-Product Method
- Constant Multiplier Technique
- Additive Congruential Method
- Linear Congruential Method

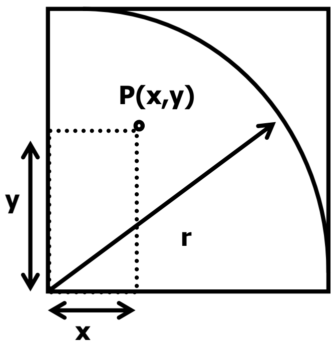
etc ….

(You are encouraged to find out the details of these methods from library.)

A Classical Application of Random Number

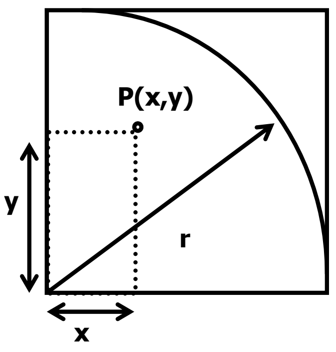
Using random numbers to compute the value of π.

Let (x, y) denote a point P in the square.
For those points on the circumference, we have
$x^2 + y^2 = r^2$.



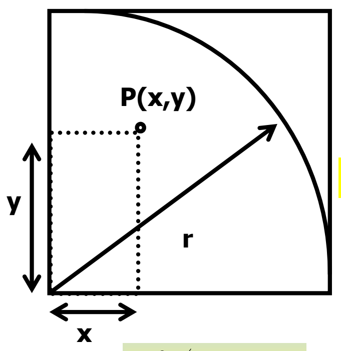
Now we cover the square by 10000 random dots.
Let n be the number of dots within the quarter circle.
By ratio, we have

$$\frac{\text{Area of quarter circle}}{\text{Area of square}} = \frac{\text{Number of dots within the quarter circle}}{\text{Number of dots within the square}}$$



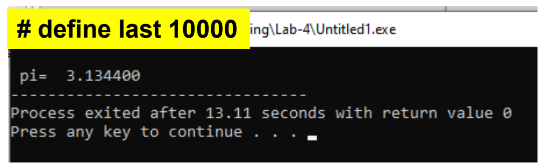
$$\frac{\pi r^2 / 4}{r * r} = \frac{n}{10000}$$
$$\frac{\pi 1^2 / 4}{1 * 1} = \frac{n}{10000}$$
$$\Rightarrow \pi = \frac{n}{2500}$$

```
// pi.c
# include <stdio.h>
# include <stdlib.h>

# define last 10000
# define seed 31

main()
{
  long n=0, i;
  double x,y;

  srand(seed); // 0 .. 32767

  for (i=0;i<last;i++)
  {
    x = rand()/32767.0;
    y = rand()/32767.0;

    if (x*x + y*y < 1.0) n++;
  }

  printf ("\n pi=  %f",n/(last/4.0));

  return 0;
}
```



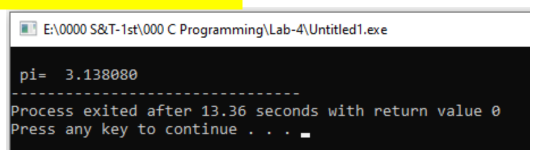
$$\frac{\pi r^2 / 4}{r * r} = \frac{n}{10000}$$
$$\frac{\pi 1^2 / 4}{1 * 1} = \frac{n}{10000}$$
$$\Rightarrow \pi = \frac{n}{2500}$$

# define last 10000



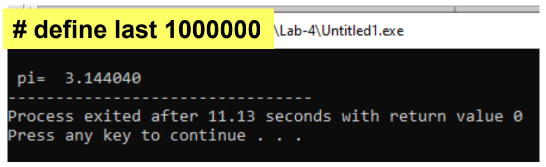
```
ing\Lab-4\Untitled1.exe

 pi=  3.134400
--------------------------------
Process exited after 13.11 seconds with return value 0
Press any key to continue . . .
```

# define last 50000



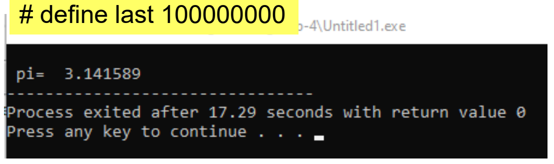
```
E:\0000 S&T-1st\000 C Programming\Lab-4\Untitled1.exe

 pi=  3.138080
--------------------------------
Process exited after 13.36 seconds with return value 0
Press any key to continue . . .
```

# define last 1000000



```
\Lab-4\Untitled1.exe

 pi=  3.144040
--------------------------------
Process exited after 11.13 seconds with return value 0
Press any key to continue . . .
```

# define last 100000000



```
-4\Untitled1.exe

 pi=  3.141589
--------------------------------
Process exited after 17.29 seconds with return value 0
Press any key to continue . . .
```