

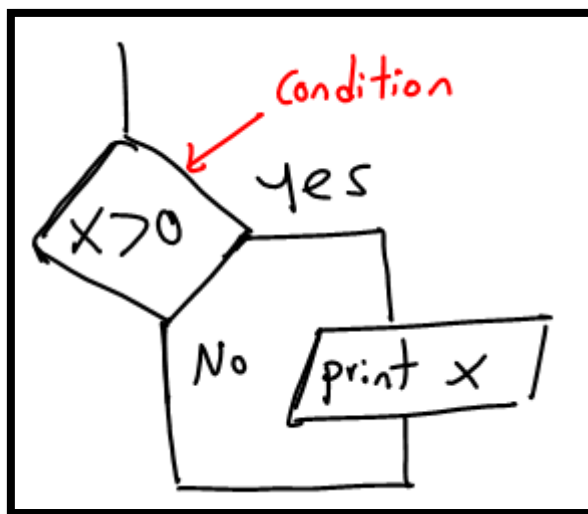
Chapter 05

Decisions

■ Decision

- We all need to be able to alter our actions in the face of changing conditions. If the forecast is for rain, then I'll take my raincoat. If the freeway is under construction, then I'll take the back road.
- Computer languages, too, must be able to perform different sets of actions depending on the conditions.
- C has three major decision-making structures
 - IF statement
 - IF-ELSE statement
 - SWITCH statement
 - Conditional operator

■ The if statement



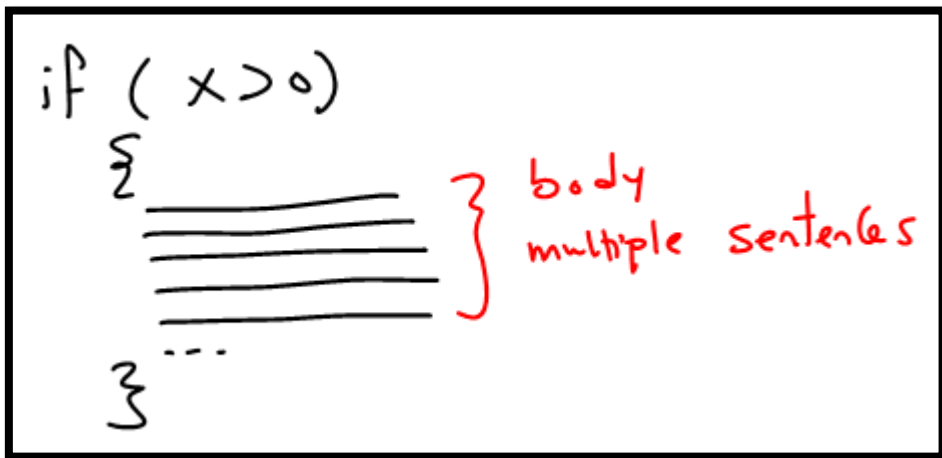
```
if (x > 0)
{
    printf("%f", x);
}
```

```
if (x > 0)
    printf("%f", x);
```

keyword Conditional expression

```
if (ch == 'y')
    printf("\n You typed y.");
```

body of if statement



- There is no "**then**" keyword in C
- The if statement is similar to the while statement in operation as well as format. In both cases the statements making up the **body of the statement won't be executed at all if the condition is false**. However, in the while statement, if the condition is true, the statement (or statements) in the body of the loop will be executed over and over until the condition becomes else; **whereas in the if statement they will be executed only once**.

IF condition

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
```

```
char ch;
printf("\nType a character: ");
scanf("%c",&ch);
if(ch == 'y')
    printf("\nYou typed y.");
}
```

Type a character: y

You typed y.

Type a character: x

Example:

Write a C program to count not only the number of characters but the number of words as well.

Word Count

```
#include <stdio.h>
#include <stdlib.h>
main()
{
    char ch = 'x';
    int cc = 0;
    int wc = 0;
    while ((ch = getche()) != '\r' )
    {
        cc++;
    }
}
```

```
        if (ch == ' ')
            wc++;

    }

    printf("\nNumber of characters: %d",cc);
    printf("\nNumber of words: %d",++wc);
}
```

```
As I said before, You are Fashel Kabeer:-)
Number of characters: 42
Number of words: 8
```

- This statement causes the variable **wc** to be incremented every time a space is detected in the input stream.
- There will always be one more word than there are spaces between them (assuming no multiple spaces), so we add 1 to the variable wordcnt before printing it.
- As in the case of the various loop statements, the body of the if statement may consist of either a **single statement terminated by a semicolon** or by a **number of statements enclosed in braces**.
-

■ Nested If statements

Nested If statement

```
#include <stdio.h>
#include <stdlib.h>
main()
{
    if (getche() == 'N')
        if (getche() == 'o')
            printf("\nYou typed No.");
}
```

- Nesting here means that one if statement is part of the body of another if statement.

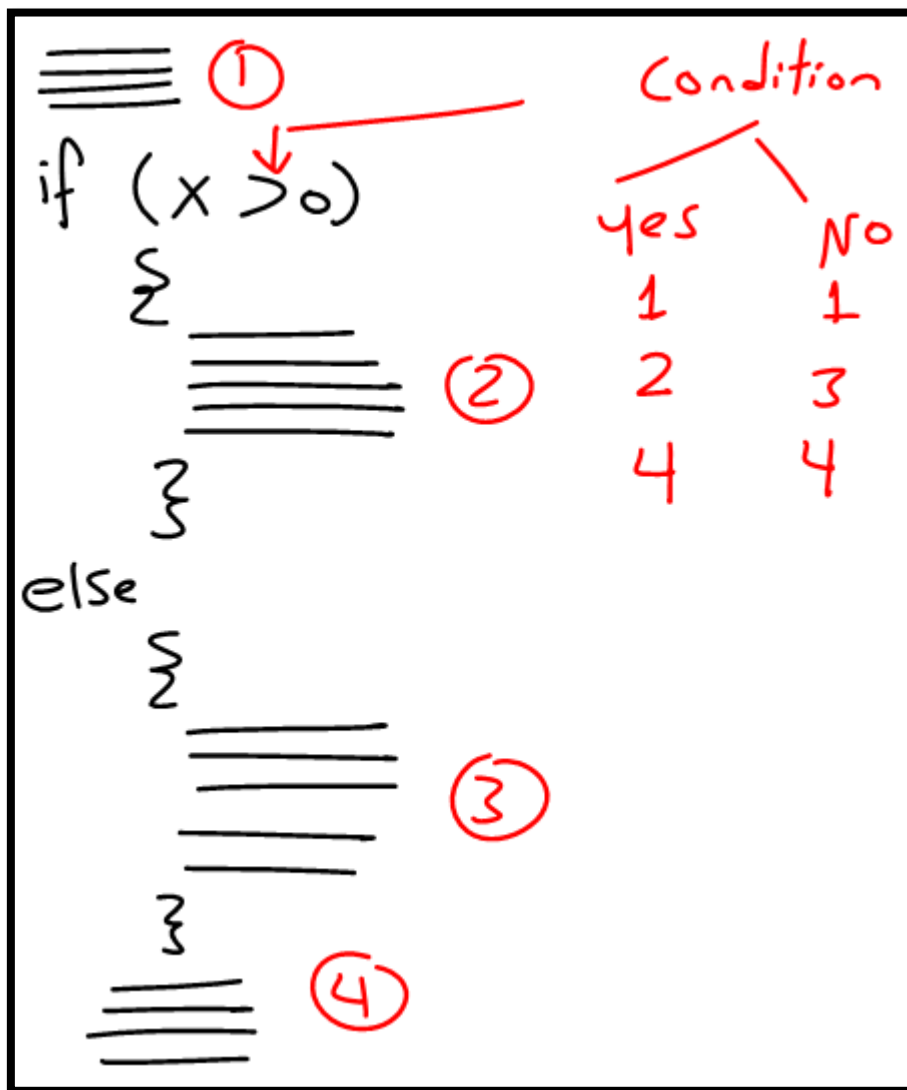
■ If-else statement

- The if statement by itself will execute a single statement, or a group of statements, when the test expression is true.
- It does nothing when it is false. **Can we execute a group of statements if and only if the test expression is not true?**
- Of course. This is the purpose of the else statement, which is demonstrated in the following example:

```
#include <stdio.h>
#include <stdlib.h>
```

```
main()
{
    char ch;
    printf("\nType a character: ");
    ch = getchar();
    if(ch == 'y')
    {
        printf("\nYou typed y.");
    }
    else
    {
        printf("\nYou did not type y");
    }
}
```

- For clarity, each **else** should be indented the same amount as its matching **if**



■ What if gets the else

What **if** gets the **else**

```
#include <stdio.h>
#include <stdlib.h>
```

```
main()
{
    int temp;
    printf("please type in the temperature: ");
    scanf("%d",&temp);
    if(temp < 28)
        if(temp >21)
            printf("Nice Day")
    else
        printf("Sure is hot");
}
```

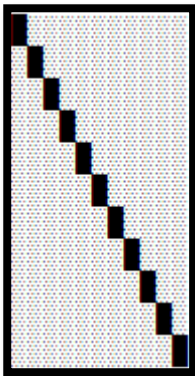
- Read the code carefully, and then run to check it
 - Print out the result in case temp equal to 23
 - Print out the result in case temp equal to 90
 - Print out the result in case temp equal to 19
- An **else** is associated with the last **if** that does not have its own **else**

■ Logical operators

- There are three logical (sometimes called "Boolean") operators in C
 - || Logical OR
 - && Logical AND
 - ! Logical NOT

Draw diagonal line

```
#include <stdio.h>
#include <stdlib.h>
main()
{
    int x,y;
    for(x = 1;x<12;x++)
    {
        for(y = 1;y<12;y++)
            if(x ==y)
                printf("\xDB");
            else
                printf("\xB0");
        printf("\n");
    }
}
```



■ Example

Count characters and numerical digits in a phrase

Count
<pre>#include <stdio.h> #include <stdlib.h> main() { int charcnt = 0; int digitcnt = 0; char ch; printf("Type in a phrase:\n"); while((ch = getche()) != '\r'); { charcnt++; if(ch>47 && ch <58) digitcnt++; } printf("\nCharacter count is %d",charcnt); printf("\nDigit count is %d",digitcnt); }</pre>

- Could you expect what is the output
- Read carefully and run this code

■ Example

Count characters and numerical digits in a phrase

Count
<pre>#include <stdio.h> #include <stdlib.h> main() { int charcnt = 0; int digitcnt = 0; char ch; printf("Type in a phrase:\n"); while((ch = getche()) != '\r') { charcnt++; if(ch>47 && ch <58) digitcnt++; } printf("\nCharacter count is %d",charcnt); printf("\nDigit count is %d",digitcnt); }</pre>

- Logical operators have lower precedence than relational operators.
- The third logical operator is the NOT operator, represented by the exclamation point (!) and sometimes called the **bang operator**. This operator reverses the logical value of the expression it operates on; it makes a true expression false and a false expression true.
- Here's an example of the NOT operator applied to a relational expression.

`! (x < 5)`

This means "not x less than five." In other words, if x is less than 5, the expression will be false, because (x < 5) is true.

We could express the same condition as

`(x >= 5)`

- The NOT operator is often used to reverse the logical value of a single variable, as in the expression

`if (! flag)`

This is more concise than the equivalent

`if (flag==0)`

■ Operator precedence

<i>Operators</i>	<i>Type</i>
! -	Unary: logical NOT, arithmetic minus.
* / %	Arithmetic (multiplicative).
+ -	Arithmetic (additive).
< > <= >=	Relational (inequality).
== !=	Relational (equality).
&&	Logical AND and OR.
= += -= *= /= %=	Assignment

■ The ELSE-IF Construct

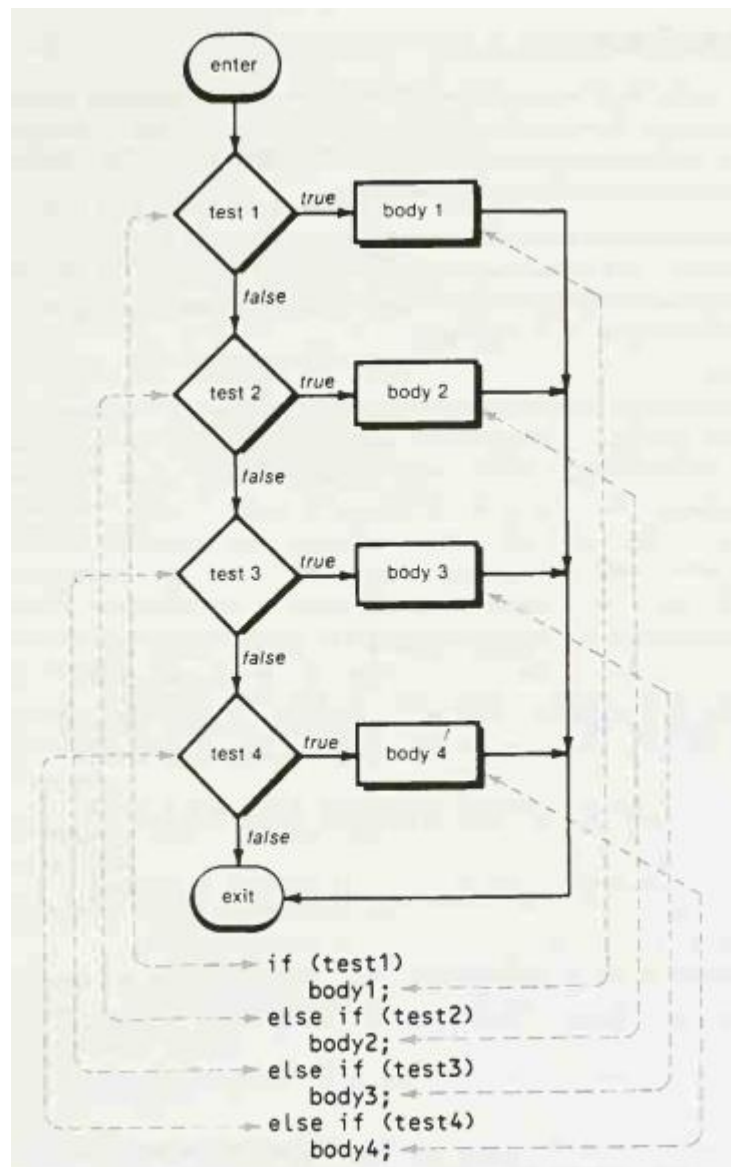
- The else -if construction is a reformatting of nested if -else statements.

Calc

```
#include <stdio.h>
#include <stdlib.h>
main()
{
    float num1=1,num2 = 1;
    char op;
    while (num2!=0)
    {
        printf("\nType number, operator,
number:\n");
        scanf("%f %c %f",&num1,&op,&num2);
```

```
        if(op == '+')
            printf("sum = %f", num1+num2);
        else if (op == '*')
            printf("Mul = %f", num1*num2);
        else if (op == '/')
            printf("Div = %f", num1/num2);
        else if (op == '-')
            printf("Sub = %f", num1-num2);
        else
            printf("invalid op");
    }
}
```

```
Type number, operator, number:
4 x 2
Sub = 2.000000
Type number, operator, number:
4*2
Mul = 8.000000
Type number, operator, number:
4 - 1
Sub = 3.000000
Type number, operator, number:
1 / 4
Div = 0.250000
Type number, operator, number:
2 / 0
Div = 1.#INF00
```



- if the test expression that follows is true, execute the statement in the body of the **else-if** (in this case a `printf()` statement) and go to the end of the entire **else-if** chain; otherwise, go to the next **else-if** statement in the chain."

■ Break statement

- **Break** is often useful when a condition suddenly occurs that makes it necessary to **leave a loop** before the loop expression becomes false.
 - As we'll see next, it is also essential in the switch statement.

```
guess
#include <stdio.h>
#include <stdlib.h>
main()
{
    float guess, incr;
    char ch;
    printf("think of a number between 1 and 99,
and\n");
    printf("I'll guess what it is. type 'e' for
equals,\n");
    printf("'g' for greater than, and 'l' for
        less than.\n");
    incr = guess = 50;
    while(incr>1.0)
    {
        printf("\nIs your number greater than
or less than %.0f\n",guess);
        incr = incr/2;
        if((ch=getche())=='e')
```

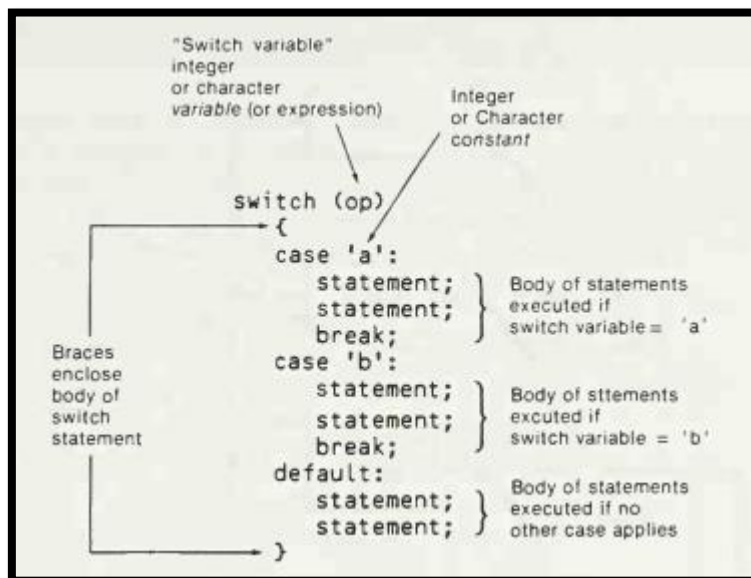
```
        break;
    else if(ch=='g')
        guess +=incr;
    else
        guess -=incr;
}
printf("\nThe number is %.0f. Am I not
clever?",guess);
}
```

```
think of a number between 1 and 99, and
I'll guess what it is. type 'e' for equals,
'g' for greater than, and 'l' for less than.
```

```
Is your number greater than or less than 50
g
Is your number greater than or less than 75
g
Is your number greater than or less than 88
l
Is your number greater than or less than 81
g
Is your number greater than or less than 84
l
Is your number greater than or less than 83
e
The number is 83. Am I not clever?
```

■ Switch

- The **switch** statement is similar to the **else -if** construct but has more flexibility and a clearer format.
- It is analogous to the case statement in Pascal and the select case statement in the newer versions of BASIC



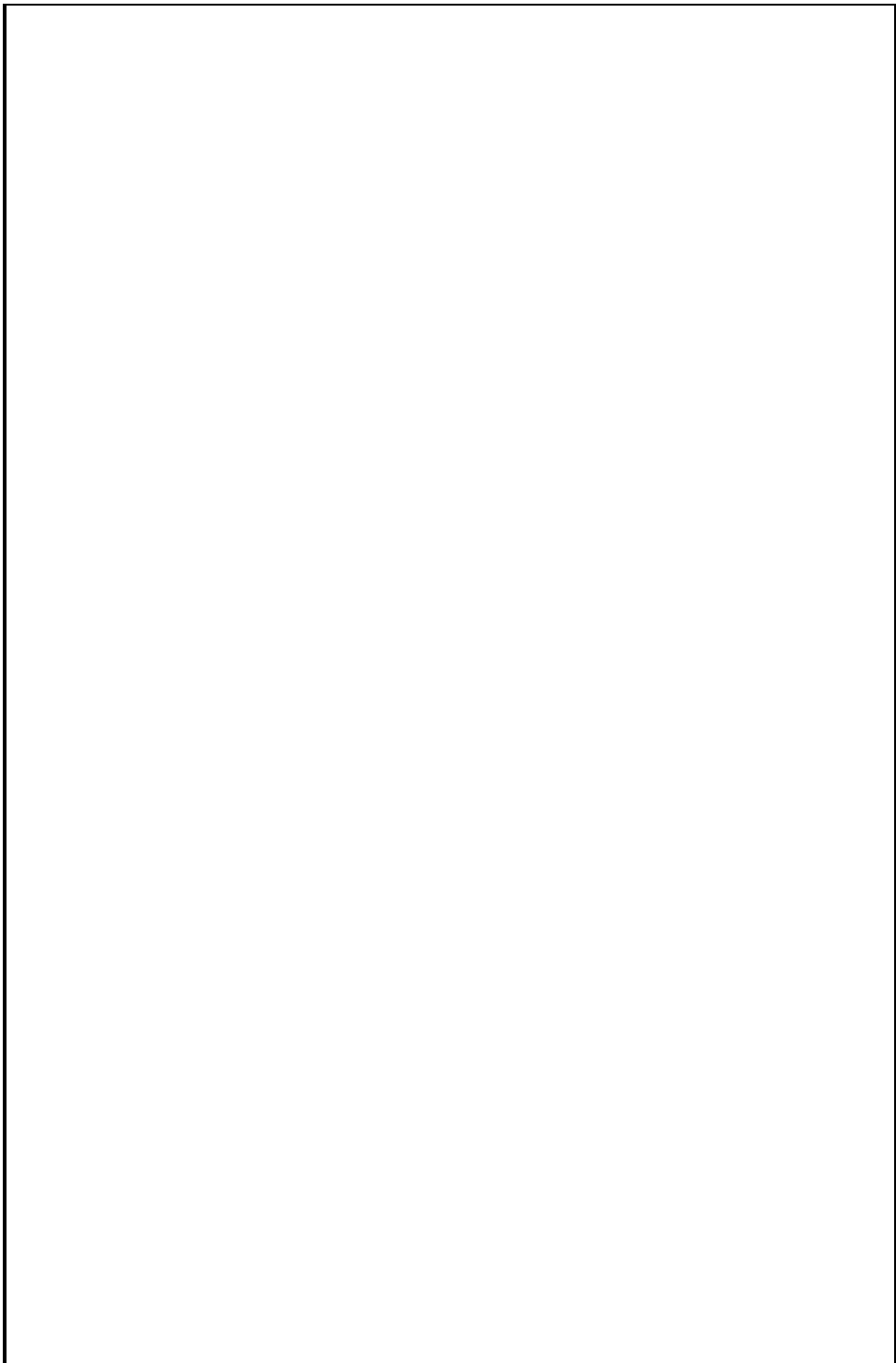
- If **no break statement is used** following a case, control will fall through to the next case.
 - Without the break, the program will execute not only the statements for a particular case but all the statements for the following cases as well.
- Structurally, the statement starts out with the keyword **switch**, followed by parentheses containing an integer or character variable which we'll call the switch variable (although it can also be an expression, like `a+b`). The structure of the switch statement is shown in Figure above.
- Following each of the case keywords is an integer or character constant. (It can be a constant expression, like `'a'` or `2`, but it must evaluate to a constant; variables aren't allowed here.)

- This constant is terminated with a colon (not a semicolon).
- There can be one or more statements following each case keyword. These statements need not be enclosed by braces, although the entire body of the switch statement—all the cases—is enclosed in braces.

■ Report

Calculator - using Switch

- | |
|---|
| <ul style="list-style-type: none">▪ Write a C program to read in two numbers and one operator, and then the program does the arithmetic function on the two numbers |
|---|



```
Type a number, operator, number:  
10 / 2
```

```
Div = 5.000000
```

```
Type a number, operator, number:  
10 * 2
```

```
Mul = 20.000000
```

```
Type a number, operator, number:  
10 - 2
```

```
Sub = 8.000000
```

```
Type a number, operator, number:  
10 + 2
```

```
Sum = 12.000000
```

```
Type a number, operator, number:
```

Sheet 03

1. Write a program that asks the user for the length, width, and height of a package, and displays "Reject" if any dimension is greater than 10 cm, and "Accept" if all the dimensions are less than or equal to 10 cm
2. Write a program that displays a counter in the middle of the window. The counter should count down from 1000 to 0, and at 0 should generate a series of beeps. Write the program so that the user can press any key during the counting to stop the counting and end the program without the beeps
3. Write a program that takes a non-negative long as input and displays each of the digits on a separate line. The program output should look similar to:

Enter an integer: 71345

7

1

3

4

5

4. An interesting (yet unsolved) question in mathematics concerns "hailstone numbers.". This series is produced by taking an initial integer, and if the number is even, dividing it by 2. If the initial number is odd, multiply it by 3 and add 1.

This process is then repeated. For example, an initial number of 10 produces:

10, 5, 16, 8, 4, 2, 1, 4, 2, 1, 4, 2, 1, ...

An initial value of 23 produces

23, 70, 35, 106, 53, 160, 80, 40, 20, 10, 5, 16, 8, 4, 2, 1, 4, 2, 1, ...

Notice that both numbers eventually reach the cycle 4, 2, 1, 4, 2, 1, ... write two programs that answer the following questions for initial values of 1 to 200

- a) Do all numbers from 1 to 200 eventually reach this cycle?
 - b) What is the maximum number of iterations taken to reach the cycle and which starting number produces the maximum?
5. "chaos theory" is a subfield of mathematics which relies heavily on the computer. A simple chaos experiment is:

Take any real number X between 0 and 1 as a starting value. Generate a new number using the "logistic" equation:

$$X = 2 * X * (1 - X)$$

Display this new X , and repeat the process. Perform this cycle 50 times

- a) Write a program that asks the user for a starting value and then performs this process. Try to determine what happens for different starting values
- b) Modify the program so that the 2 in the equation can be replaced with a value specified by the user in the range 2 to 4, but the starting value of X is always 0.5. see if you can find interesting behavior