C Program Controls + Flow Charts

-controlling the program execution flow: selection, repetition and branching-

PROGRAM CONTROL

- Program begins execution at the `main()` function.
- Statements within the `main()` function are then executed from top-down style, line-by-line.
- However, this order is rarely encountered in real C program.
- The order of the execution within the `main()` body may be branched.
- Changing the order in which statements are executed is called program control.
- Accomplished by using program control statements.
- So we can control the program flows.

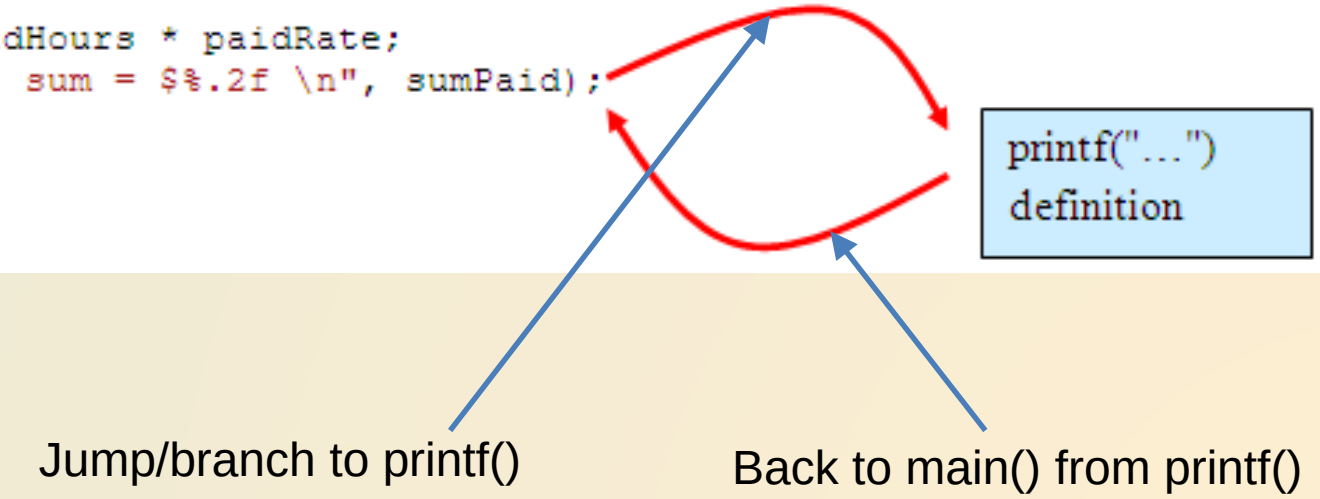
PROGRAM CONTROL

- There are three types of program controls:
  1. Sequence control structure.
  2. Selection structures such as `if`, `if-else`, nested `if`, `if-if-else`, `if-else-if` and `switch-case-break`.
  3. Repetition (loop) such as `for`, `while` and `do-while`.
- Certain C functions and keywords also can be used to control the program flows.

PROGRAM CONTROL

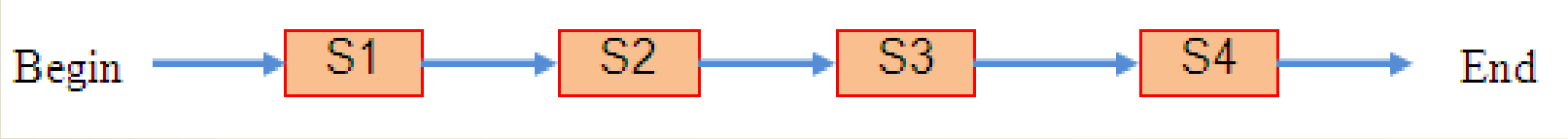
- Take a look at the following example

```
#include <stdio.h> // put stdio.h file here

int main(void)
{
    float paidRate = 5.0, sumPaid, paidHours = 25;

    sumPaid = paidHours * paidRate;
    printf("Paid  sum = $%.2f \n", sumPaid);

    return 0;
}
```



PROGRAM CONTROL

`float paidRate = 5.0, sumPaid, paidHours = 25;`	S1
`sumPaid = paidHours * paidRate;`	S2
`printf("Paid  sum = $%.2f \n", sumPaid);`	S3
`return 0;`	S4



- One entry point and one exit point.
- Conceptually, a control structure like this means a sequence execution.

PROGRAM CONTROL

Selection Control Structure

- Program need to select from the options given for execution.
- At least 2 options, can be more than 2.
- Option selected based on the *condition* evaluation result: TRUE or FALSE.

PROGRAM CONTROL

Selection: if, if-else, if-else-if

- Starting from the most basic if syntax,

`if (condition)`	`if (condition)`
`  statement;`	`  { statements;}`
`next_statement;`	`next_statement;`

1. `(condition)` is evaluated.
2. If TRUE (non-zero) the `statement` is executed.
3. If FALSE (zero) the `next_statement` following the `if` statement block is executed.
4. So, during the execution, based on some condition, some codes were skipped.

PROGRAM CONTROL

For example:

```
if (hours > 70)
    hours = hours + 100;
printf("Less hours, no bonus!\n");
```

- If `hours` is less than or equal to 70, its value will remain unchanged and the `printf()` will be executed.
- If it exceeds 70, its value will be increased by 100.

```
if(jobCode == '1')
{
    carAllowance = 100.00;
    housingAllowance = 500.00;
    entertainmentAllowance = 300.00;
}
printf("Not qualified for car, housing and entertainment allowances!");
```

The three statements enclosed in the curly braces { } will only be executed if `jobCode` is equal to '1', else the `printf()` will be executed.

PROGRAM CONTROL

`if (condition)`	`if (condition)`
`    statement_1;`	`    { a block of statements;}`
`else`	`else`
`    statement_2;`	`    { a block of statements;}`
`next_statement;`	`next_statement;`

Explanation:

1. The (`condition`) is evaluated.
2. If it evaluates to non-zero (TRUE), `statement_1` is executed, otherwise, if it evaluates to zero (FALSE), `statement_2` is executed.
3. They are mutually exclusive, meaning, either `statement_1` is executed or `statement_2`, but not both.
4. `statements_1` and `statements_2` can be a block of codes and must be put in curly braces.

PROGRAM CONTROL

For example:

```
if(myCode == '1')
   rate = 7.20;
else
   rate = 12.50;
```

If `myCode` is *equal* to '1', the `rate` is 7.20 else, if `myCode` is *not equal* to '1' the `rate` is 12.50.

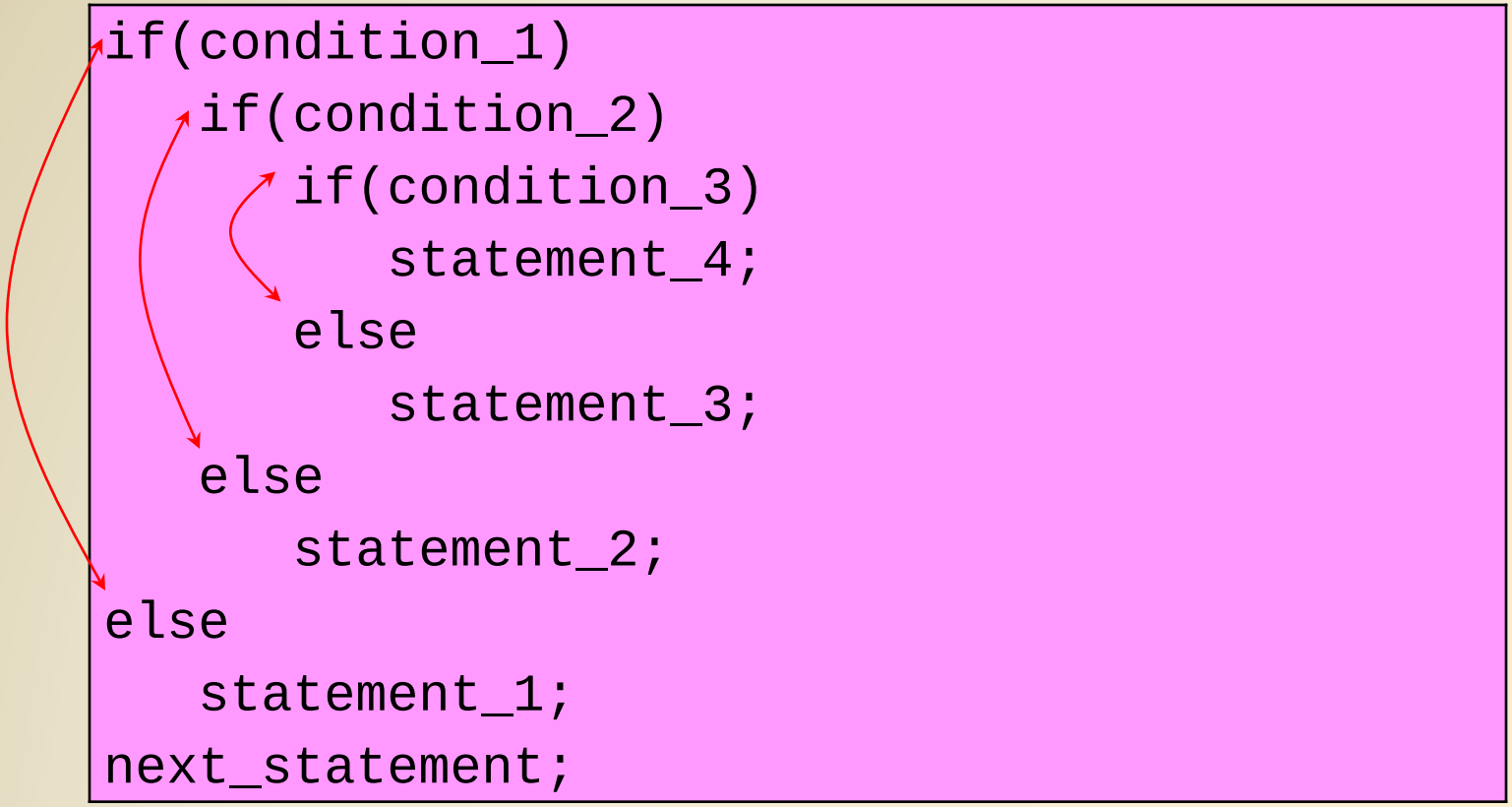
Equal/not equal (=) is not a value comparison, but a character comparison!!!

PROGRAM CONTROL

- Program example 1: if
- Program example 2: if-if
- Program example 3: if-else

PROGRAM CONTROL

- The if-else constructs can be nested (placed one within another) to any depth.
- General forms: if-if-else and if-else-if.
- The if-if-else constructs has the following form (3 level of depth example),



```
if(condition_1)
   if(condition_2)
      if(condition_3)
         statement_4;
      else
         statement_3;
   else
      statement_2;
else
   statement_1;
next_statement;
```

PROGRAM CONTROL

- In this nested form, `condition_1` is evaluated. If it is zero (FALSE), `statement_1` is executed and the entire nested if statement is terminated.
- If non-zero (TRUE), control goes to the second if (within the first if) and `condition_2` is evaluated.
- If it is zero (FALSE), `statement_2` is executed; if not, control goes to the third if (within the second if) and `condition_3` is evaluated.
- If it is zero (FALSE), `statement_3` is executed; if not, `statement_4` is executed. The `statement_4` (inner most) will only be executed if all the if statement are TRUE.
- Again, only one of the statements is executed other will be skipped.
- If the else is used together with if, always match an else with the nearest if before the else.
- *`statements_x`* can be a block of codes and must be put in curly braces.

Program example: nested if-else

PROGRAM CONTROL

- The `if-else-if` statement has the following form (3 levels example).

```
if(condition_1)
   statement_1;
else if (condition_2)
   statement_2;
else if(condition_3)
   statement_3;
else
   statement_4;
next_statement;
```

PROGRAM CONTROL

- `condition_1` is first evaluated. If it is non zero (TRUE), `statement_1` is executed and the whole statement terminated and the execution is continue on the `next_statement`.
- If `condition_1` is zero (FALSE), control passes to the next else-if and `condition_2` is evaluated.
- If it is non zero (TRUE), `statement_2` is executed and the whole system is terminated. If it is zero (FALSE), the next else-if is tested.
- If `condition_3` is non zero (TRUE), `statement_3` is executed; if not, `statement_4` is executed.
- Note that only one of the statements will be executed, others will be skipped.
- *`statement_x`* can be a block of statement and must be put in curly braces.

PROGRAM CONTROL

The if-else-if program example

- If mark is less than 40 then grade 'F' will be displayed; if it is greater than or equal to 40 but less than 50, then grade 'E' is displayed.
- The test continues for grades 'D', 'C', and 'B'.
- Finally, if mark is greater than or equal to 80, then grade 'A' is displayed.

PROGRAM CONTROL

Selection: The `switch-case-break`

- The most flexible selection program control.
- Enables the program to execute different statements based on an condition or expression that can have more than two values.
- Also called multiple choice statements.
- The if statement were limited to evaluating an expression that could have only two logical values: TRUE or FALSE.
- If more than two values, have to use nested if.
- The `switch` statement makes such nesting unnecessary.
- Used together with `case` and `break`.

PROGRAM CONTROL

- The `switch` constructs has the following form:

```
switch(condition)
{
    case  template_1  : statement(s);
                break;
    case  template_2  : statement(s);
                break;
    case  template_3  : statement(s);
                break;
    …
    …
    case  template_n : statement(s);
                break;

    default : statement(s);
}
next_statement;
```

PROGRAM CONTROL

- Evaluates the `(condition)` and compares its value with the templates following each `case` label.
- If a match is found between `(condition)` and one of the templates, execution is transferred to the `statement(s)` that follows the `case` label.
- If no match is found, execution is transferred to the `statement(s)` following the optional `default` label.
- If no match is found and there is no `default` label, execution passes to the first statement following the `switch` statement closing brace which is the `next_statement`.
- To ensure that only the statements associated with the matching template are executed, include a `break` keyword where needed, which terminates the entire `switch` statement.
- The `statement(s)` can be a block of code in curly braces.

The C switch-case-break program example

PROGRAM CONTROL

- The statement sequence for case may also be NULL or empty.
- NULL/empty switch-case-break statement example
- The program would display,

`B stands for Blue colour!`	If the value entered at the prompt is B;
`You have chosen 'G', 'R' or 'Y'` `G stands for Green, R for Red and Y for Yellow!`	If the value entered at the prompt is G or R or Y;
`The initial not a chosen colour!`	If there is no matching characters.

- It is useful for multiple cases that need the same processing sequence.

PROGRAM CONTROL

- The `break` statement may be omitted to allow the execution to continue to the next cases.

The switch-case-break without break program example

- It will display the message "`Choice number 1!`" if `nChoice == 1`.
- It will display the message "`Choice number 2!`" if `nChoice == 2`.
- It will display both the messages "`Choice number 3!`" and "`Choice number 4!`" if `nChoice == 3`.
- It will display the "`Invalid choice!`" if it has any other value.
- The `switch-case` construct can also be nested.

PROGRAM CONTROL

- The differences between nested if and switch:

  1. The `switch-case` permits the execution of more than one alternatives (by not placing `break`) whereas the `if` statement does not. In other words, alternatives in an `if` statement are mutually exclusive whereas they may or may not be in the case of a `switch-case`.
  2. A `switch` can only perform equality tests involving integer (or character) constants, whereas the `if` statement allows more general comparison involving other data types as well.

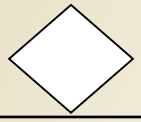
- When there are more than 3 or 4 conditions, use the `switch-case-break` statement rather than a long nested `if` statement.
- When there are multiple options to choose from.
- When test condition only use integer (or character) constants.

PROGRAM CONTROL

A flow-chart story

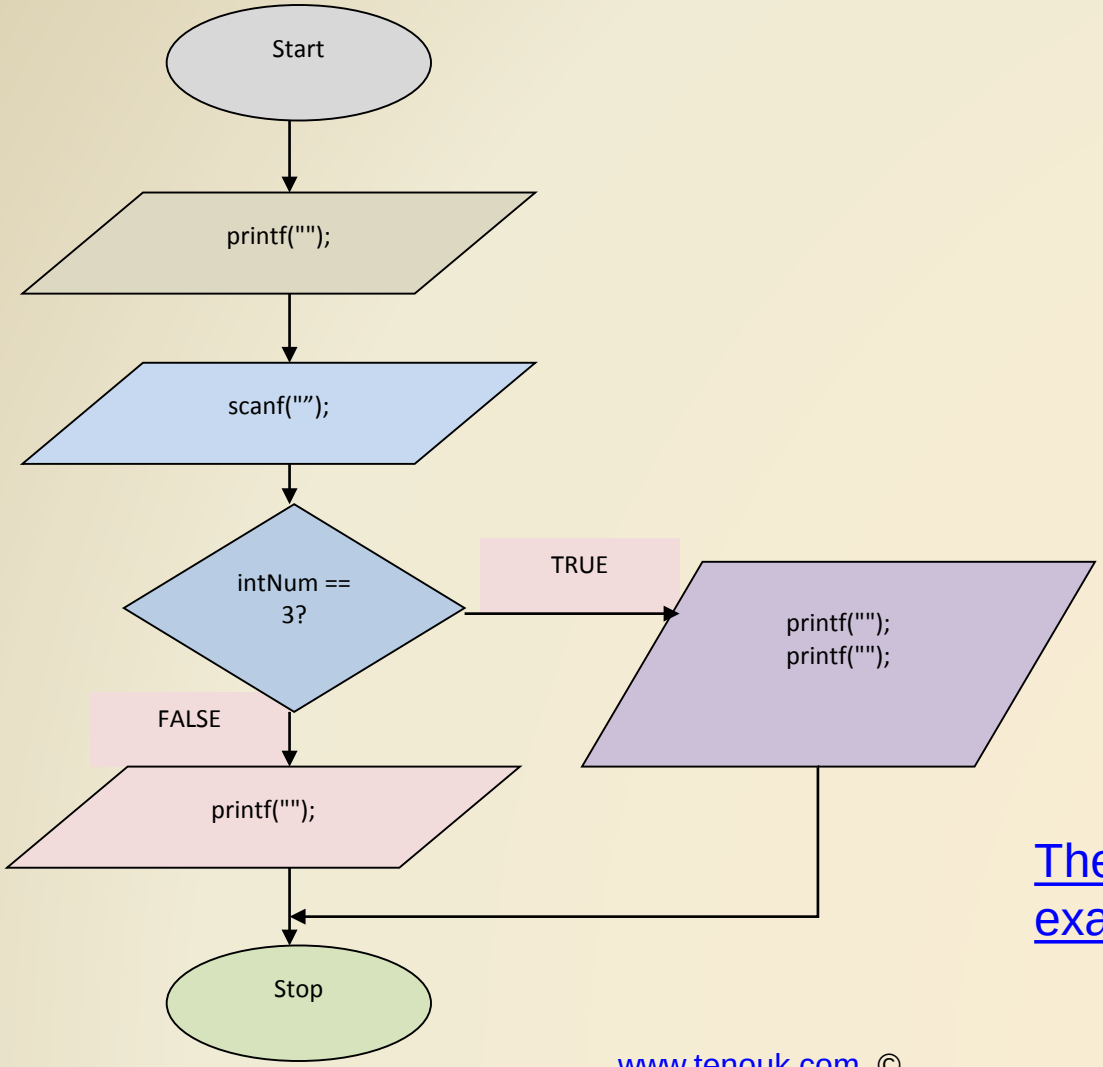
- A graphical representation of an algorithm.
- Drawn using certain symbols such as rectangles, diamonds, ovals, and small circles.
- These symbols are connected by arrows called flow lines.
- Flow-charts clearly show the program's execution order and indirectly describe how control structures operate.

PROGRAM CONTROL

Symbol	Name	Description
	Rectangular or action	A process or an action such as calculation and assignment
	Oval	Begin/start or End/stop. Indicates a completed algorithm or program flow
	Diamond or decision	Indicates a decision to be made such as YES/NO, TRUE/FALSE, <, <= etc.
	Flow lines	Indicates the order of the actions to be executed, connecting other symbols
	Small circle or connector	Indicates a portion of a complete algorithm continued from the previous portion or to be continued to the next portion
	Input or output	The input or output such as standard input or output

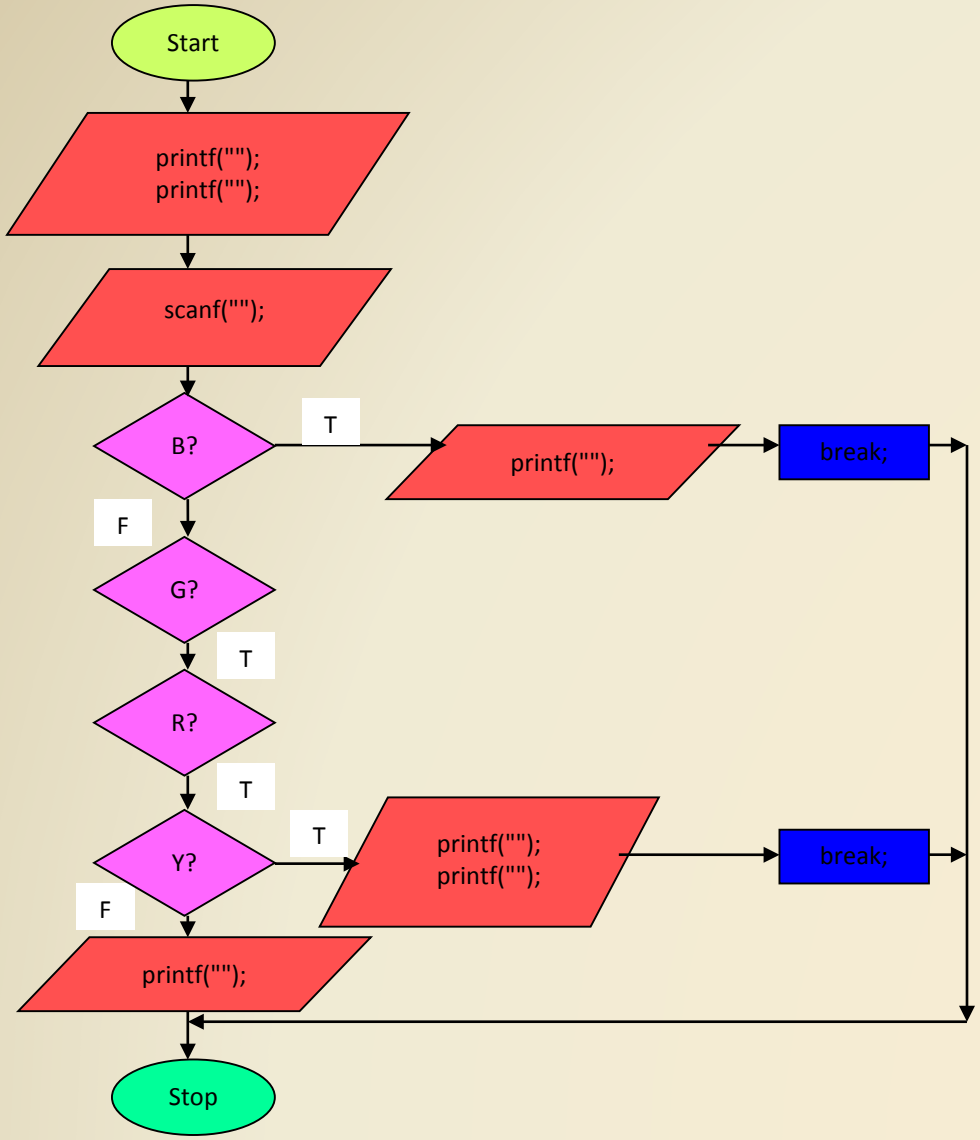
PROGRAM CONTROL

- The following flow chart examples represent C `if` selection constructs.



The if program source code example describing flow chart

PROGRAM CONTROL

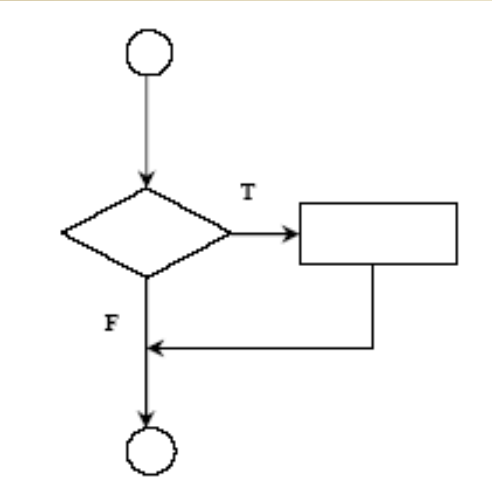


- The following flow chart examples represent C switch-case selection constructs.

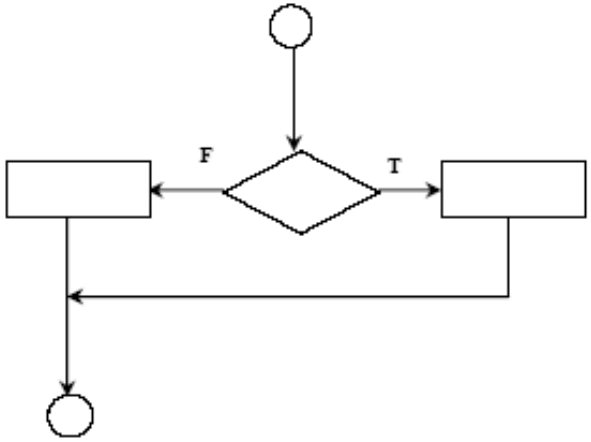
[The switch-case-break program source code example]

PROGRAM CONTROL

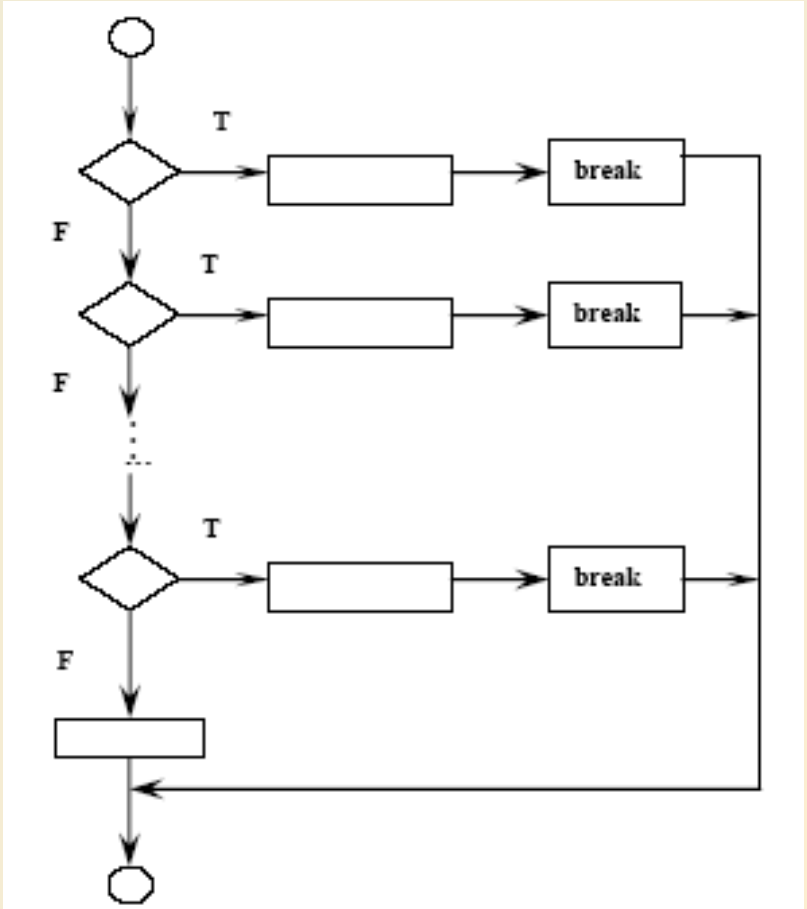
- `if`, `if-else` and `switch-case-break` flow charts



`if` structure – single selection



`if-else` structure – double selection



`switch` structure – multiple selections

PROGRAM CONTROL

Repetition: The `for` statement

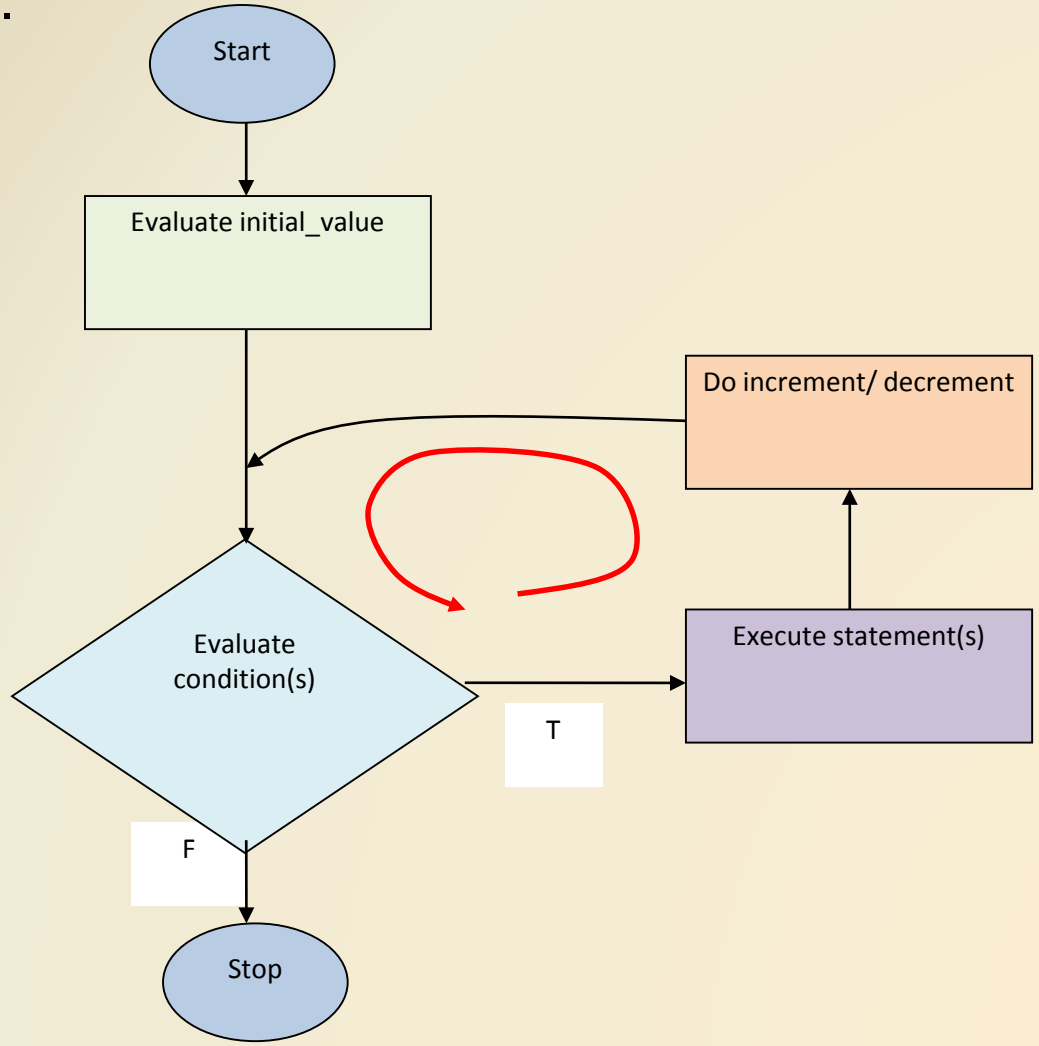
- Executes a code block for a certain number of times.
- The code block may have no statement, one statement or more.
- The `for` statement causes the `for` loop to be executed in a fixed number of times.
- The following is the `for` statement form,

```
for(initial_value;condition(s);increment/decrement)
    statement(s);
next_statement;
```

- `initial_value`, `condition(s)` and `increment/decrement` are any valid C expressions.
- The `statement(s)` may be a single or compound C statement (a block of code).
- When `for` statement is encountered during program execution, the following events occurs:
  1. The `initial_value` is evaluated e.g. `intNum = 1`.
  2. Then the `condition(s)` is evaluated, typically a relational expression.
  3. If `condition(s)` evaluates to FALSE (zero), the `for` statement terminates and execution passes to `next_statement`.
  4. If `condition(s)` evaluates as TRUE (non zero), the `statement(s)` is executed.
  5. Next, `increment/decrement` is executed, and execution returns to step no. 2 until `condition(s)` becomes FALSE.

PROGRAM CONTROL

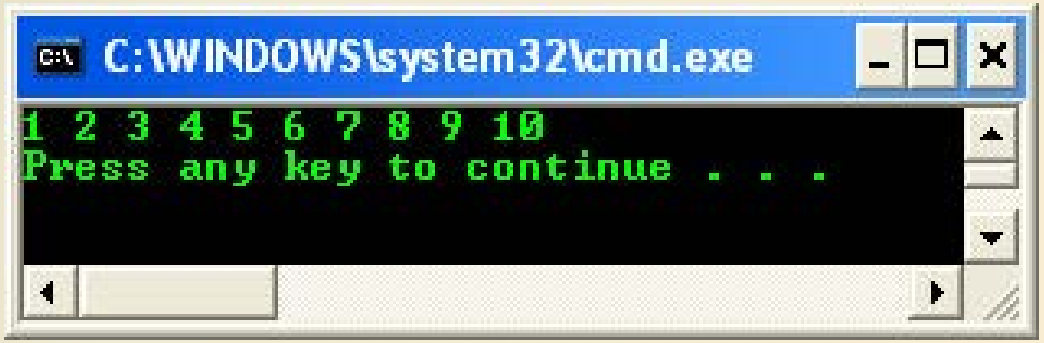
- The `for` loop flow chart should be something like the following.



PROGRAM CONTROL

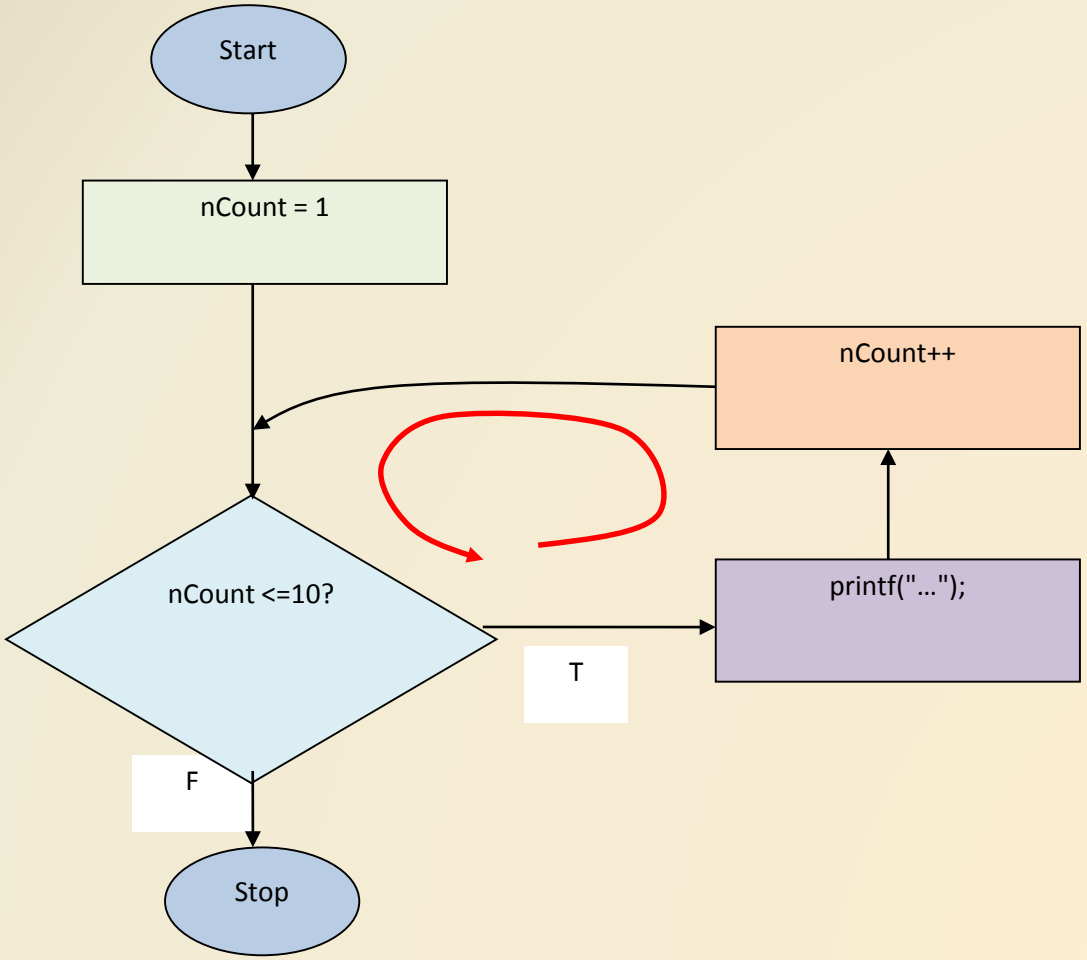
- A Simple `for` example, printing integer 1 to 10.

```
#include <stdio.h>
void main(void)
{
        int nCount;
        // display the numbers 1 to 10
        for(nCount = 1; nCount <= 10; nCount++)
                printf("%d ", nCount);
        printf("\n");
}
```



PROGRAM CONTROL

- Its flow chart…



PROGRAM CONTROL

- `for` loop is a very flexible construct.
- Can use the decrementing counter instead of incrementing. For example,

```
for (nCount = 100; nCount > 0; nCount--)
```

- Can use counter other than 1, for example 3,

```
for(nCount = 0; nCount < 1000; nCount += 3)
```

- `initial_value` can be omitted if the test variable has been initialized beforehand.
- However the semicolon must still be there. For example,

```
nCount=1;
for( ; nCount < 1000; nCount ++)
```

PROGRAM CONTROL

- The `initial_value` can be any valid C expression, the expression is executed once when the `for` statement is first reached. For example,

```
nCount =1;
for(printf("Now sorting the array…"); nCount < 1000;
    nCount ++)
```

- The increment/decrement expression can be omitted as long as the counter variable is updated within the body of the `for` statement.
- The semicolon still must be included. For example,

```
for(nCount =0; nCount < 100; )
printf("%d", nCount ++);
```

PROGRAM CONTROL

- The `condition(s)` expression that terminates the loop can be any valid C expression.
- As long as it evaluates as TRUE (non zero), the `for` statement continues to execute.
- Logical operators can be used to construct more complex `condition(s)` expressions. For example,

```
for(nCount =0; nCount < 1000 && name[nCount] != 0; nCount ++)
    printf("%d", name[nCount]);
for(nCount = 0; nCount < 1000 && list[nCount];)
    printf("%d", list[nCount ++]);
```

- Note: The `for` statement(s) and arrays are closely related, so it is difficult to define one without explaining the other (will be discussed in another Chapter).

PROGRAM CONTROL

- The `for` statement(s) can be followed by a null (empty) statement, so that task is done in the `for` loop itself.
- Null statement consists of a semicolon alone on a line. For example,

```
for(count = 0; count < 20000; count++)
    ;
```

- This statement provides a pause (delay) of 20,000 milliseconds.

PROGRAM CONTROL

- An expression can be created by separating two sub expressions with the comma operator, and are evaluated (in left-to-right order), and the entire expression evaluates to the value of the right sub expression.
- Each part of the `for` statement can be made to perform multiple duties. For example,

  "We have two arrays with 1000 elements each, named a[ ] and b[ ]. Then we want to copy the contents of a[ ] to b[ ] in the reverse order, so, after the copy operation, the array content should be…"

  `b[0], b[1], b[2],…` and `a[999], a[998], a[997],…` and so on.

- sample coding is,

```
for(iRow = 0, jColumn = 999; iRow < 1000; iRow ++, jColumn--)
    b[jColumn] = a[iRow];
```

PROGRAM CONTROL

- Another examples of the `for` statements,

```
nSum = 0;
for(iRow = 1; iRow <=20; iRow++)
   nSum = nSum + iRow;
printf("\n Sum of the first 20 natural numbers = ");
printf("Sum =  %d", nSum);
```

- The above program segment will compute and display the sum of the first 20 natural numbers.
- The above example can be re-written as,

```
for(iNum = 1, nSum = 0; iNum <= 20; iNum++)
    nSum = nSum + iNum;
printf("Sum of the first 20 natural numbers = %d", nSum);
```

- Take note that the initialization part has two statements separated by a comma (,).

PROGRAM CONTROL

- Another example,

```
for(iNum = 2, nSum=0, nSum2 = 0; iNum <= 20; iNum = iNum + 2)
{
   nSum = nSum + iNum;
   nSum2 = nSum2 + iNum * iNum;
}
 printf("Sum of the first 20 even natural numbers = %d\n", nSum);
 printf("Sum of the square for the first 20 even natural numbers =
%d", nSum2);
```

- In this example, the `for` statement is a compound or block statement.
- Note that, the initial value in the initialization part doesn't have to be zero and the increment value unnecessarily needs to be 1.

PROGRAM CONTROL

- We can also create an infinite or never-ending loop by omitting all the expressions or by using a non-zero constant for condition(s) as shown in the following two code snippets,

```
for( ; ; )
    printf("This is an infinite loop\n");
```

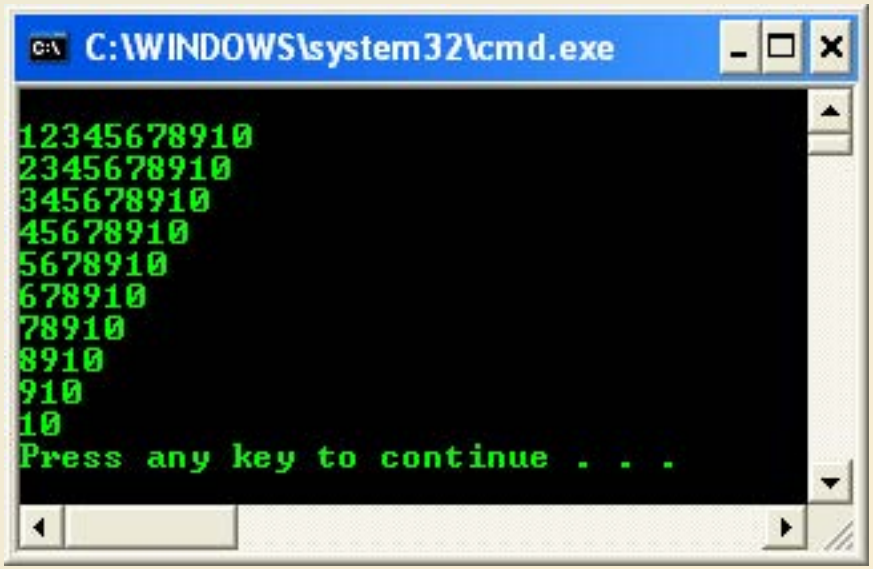
- or

```
for( ; 1 ; )
    printf("This is an infinite loop\n");
```

- In both cases, the message "`This is an infinite loop`" will be printed repeatedly, indefinitely.
- All the repetition constructs discussed so far can be nested to any degree.

PROGRAM CONTROL

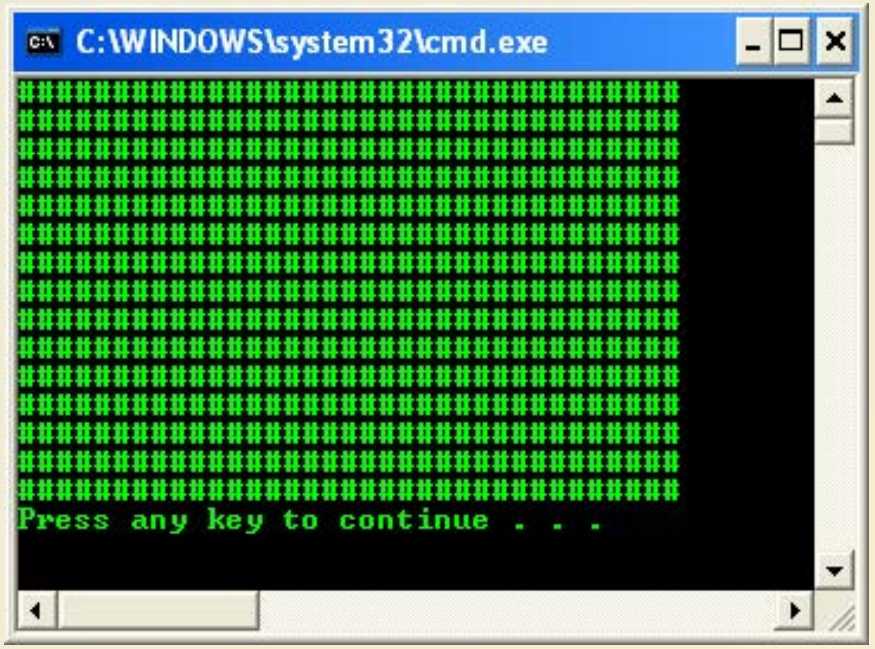
The nested for example



- The program has two `for` loops. The loop index `iRow` for the outer (first) loop runs from 1 to 10 and for each value of `iRow`, the loop index `jColumn` for the inner loop runs from `iRow + 1` to `10`.
- Note that for the last value of `iRow` (i.e. `10`), the inner loop is not executed at all because the starting value of `jColumn` is 2 and the expression `jColumn < 11` yields the value false (`jColumn = 11`).

PROGRAM CONTROL

Another nested for example



1. In the first `for` loop, the initialization is skipped because the initial value of `row`, `10` has been initialized; this `for` loop is executed until the `row` is `1` (`row > 0`).
2. For every `row` value, the inner `for` loop will be executed until `col = 1` (`col > 0`).
3. So the external `for` loop will print the `row` and the internal `for` loop will print the `column` so we got a rectangle of #.

PROGRAM CONTROL

Repetition: The `while` loop

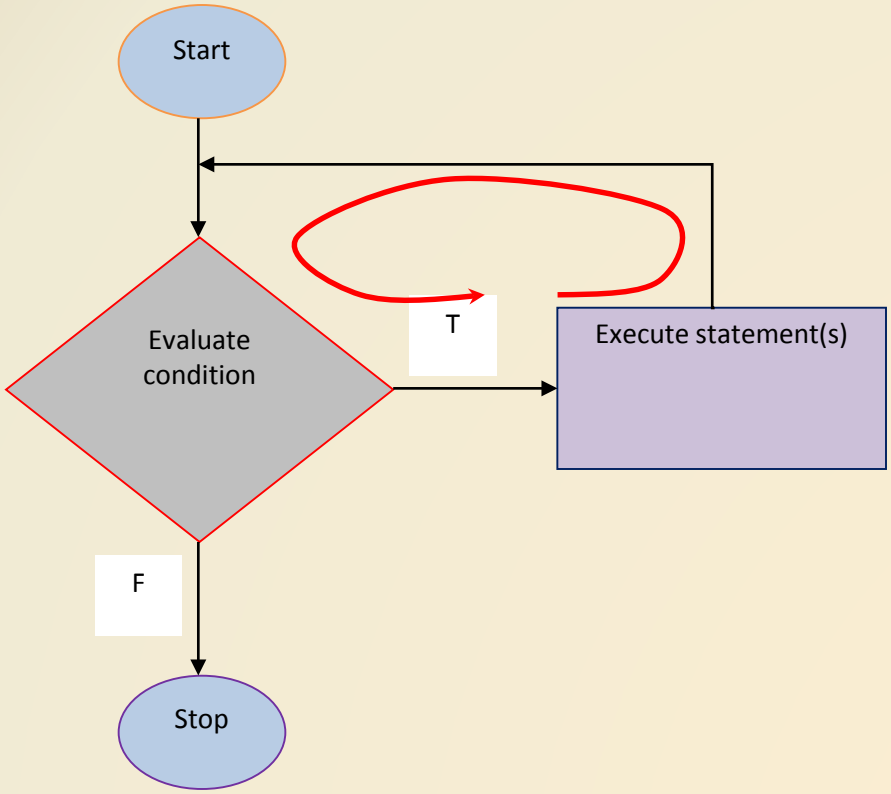
- Executes a block of statements as long as a specified condition is TRUE.
- The general `while` loop construct,

```
while (condition)
   statement(s);
next_statement;
```

- The `(condition)` may be any valid C expression.
- The `statement(s)` may be either a single or a compound (a block of code) C statement.
- When `while` statement encountered, the following events occur:
  1. The `(condition)` is evaluated.
  2. If `(condition)` evaluates to FALSE (zero), the `while` loop terminates and execution passes to the `next_statement`.
  3. If `(condition)` evaluates as TRUE (non zero), the C `statement(s)` is executed.
  4. Then, the execution returns to step number 1 until condition becomes FALSE.

PROGRAM CONTROL

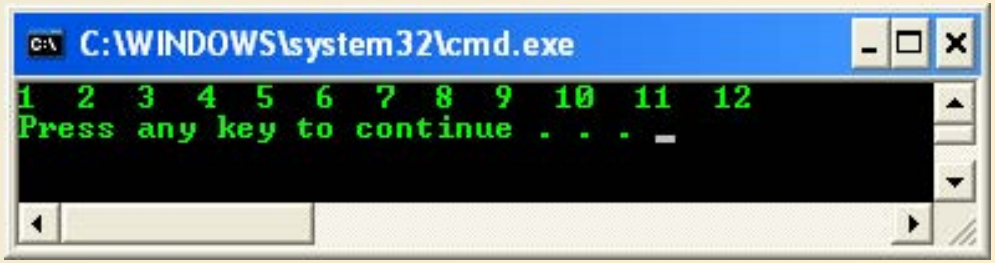
- The `while` statement flow chart is shown below.



PROGRAM CONTROL

- A simple example

```
// simple while loop example
#include <stdio.h>
int main(void)
{
        int  nCalculate = 1;
        // set the while condition
        while(nCalculate <= 12)
        {
                // print
                printf("%d  ", nCalculate);
                // increment by 1, repeats
                nCalculate++;
        }
            // a newline
        printf("\n");
        return 0;
}
```



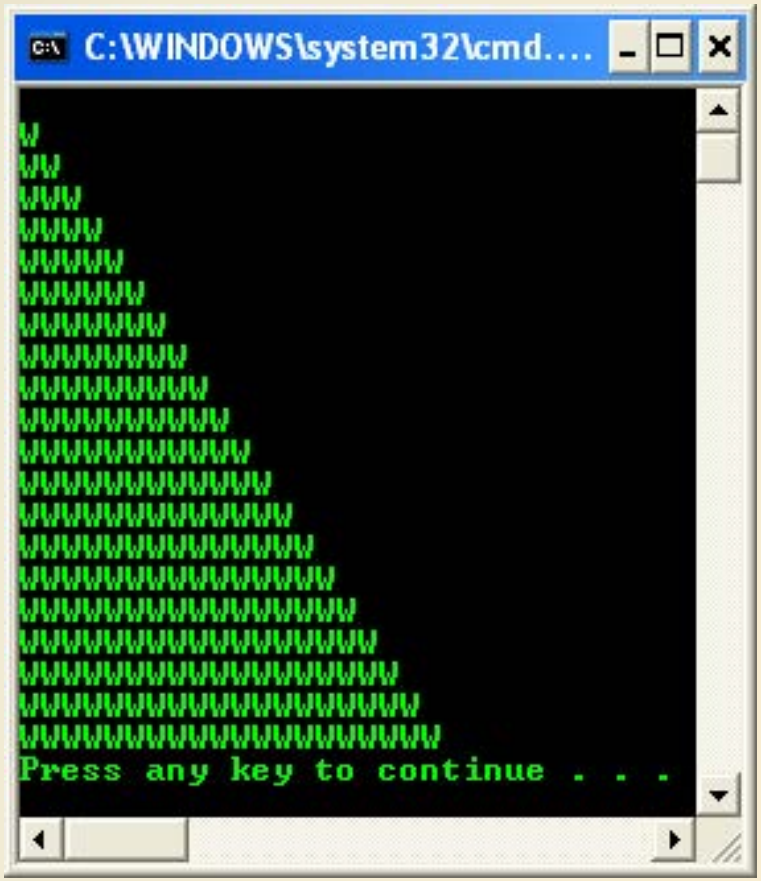
PROGRAM CONTROL

- The same task that can be performed using the `for` statement.
- But, `while` statement does not contain an initialization section, the program must explicitly initialize any variables beforehand.
- As conclusion, `while` statement is essentially a `for` statement without the initialization and increment components.
- The syntax comparison between `for` and `while`,

`for( ; condition; )`	vs	`while(condition)`

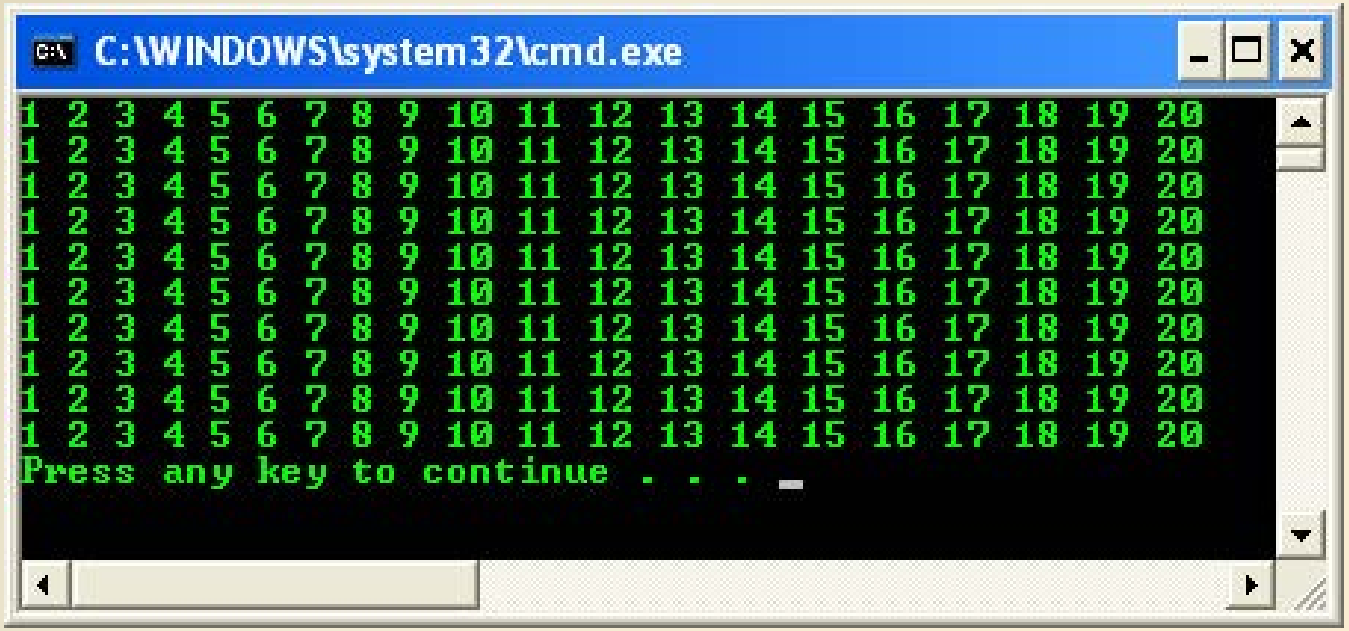
PROGRAM CONTROL

- Just like for and if statements, while statements can also be nested.
- The nested while example



PROGRAM CONTROL

- The nested for and while program example



PROGRAM CONTROL

Repetition: The `do-while` loop

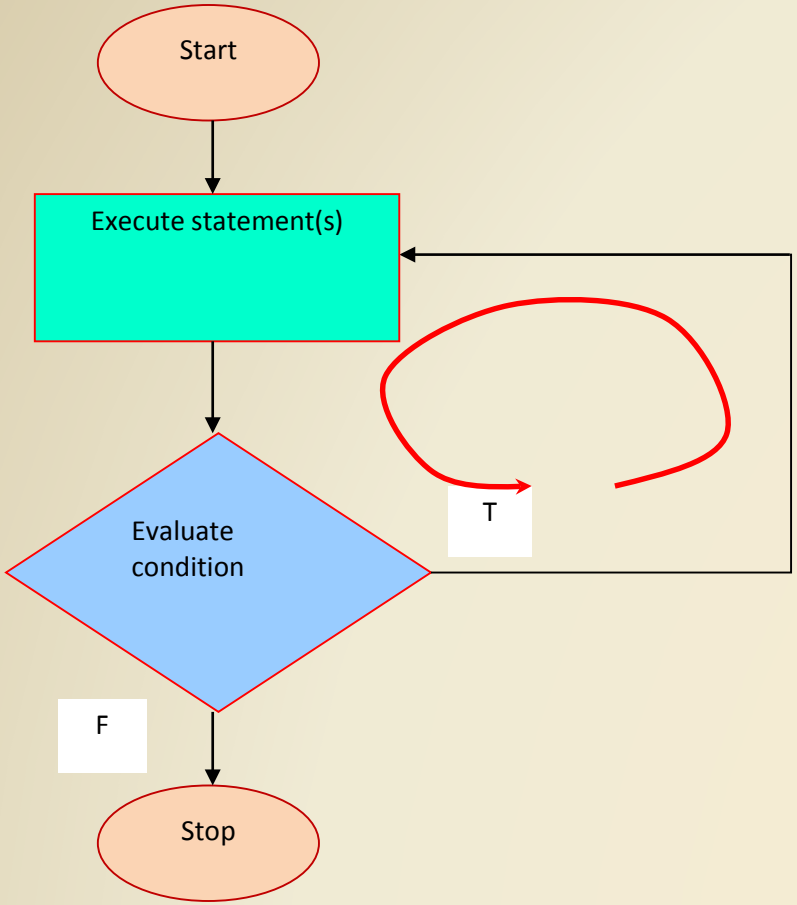
- Executes a block of statements as long as a specified condition is true at least once.
- Test the condition at the end of the loop rather than at the beginning, as demonstrated by the `for` and `while` loops.
- The `do-while` loop construct is,

```
do
    statement(s);
while (condition)
next_statement;
```

- `(condition)` can be any valid C expression.
- `statement(s)` can be either a single or compound (a block of code) C statement.
- When the program encounter the `do-while` loop, the following events occur:
  1. The `statement(s)` are executed.
  2. The `(condition)` is evaluated. If it is TRUE, execution returns to step number 1. If it is FALSE, the loop terminates and the `next_statement` is executed.
  3. This means the `statement(s)` in the `do-while` will be executed at least once.

PROGRAM CONTROL

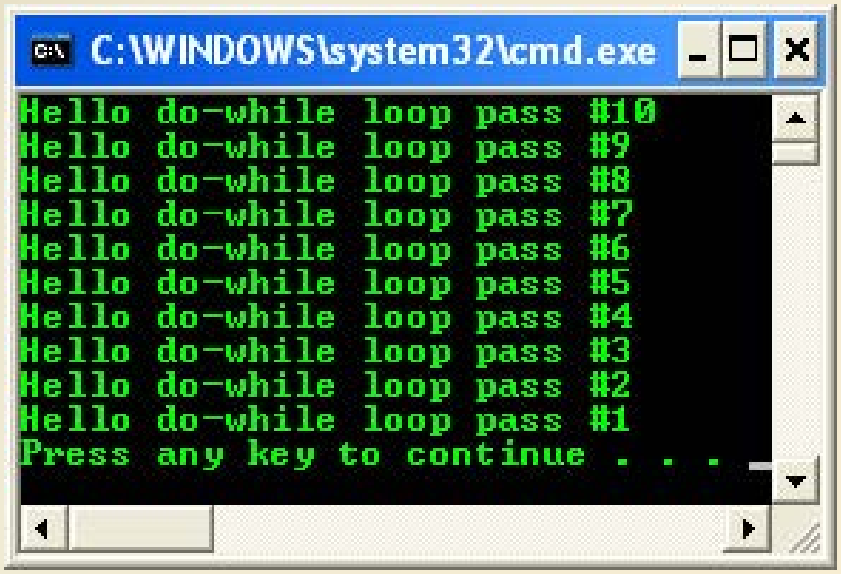
- A flow chart for the `do-while` loop



- The `statement(s)` are always executed at least once.
- `for` and `while` loops evaluate the condition at the start of the loop, so the associated statements are not executed if the condition is initially FALSE.

PROGRAM CONTROL

- The do-while program example



PROGRAM CONTROL

Other Program Controls

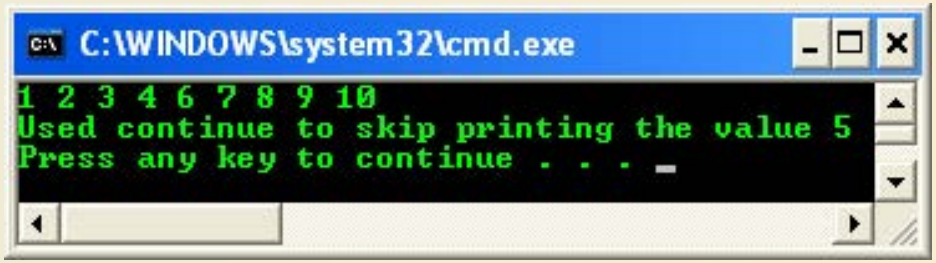
`continue` keyword

- `continue` keyword forces the next iteration to take place immediately, skipping any instructions that may follow it.
- The `continue` statement can only be used inside a loop (`for`, `do-while` and `while`) and not inside a `switch-case` selection.
- When executed, it transfers control to the condition (the expression part) in a `while` or `do-while` loop, and to the increment expression in a `for` loop.
- Unlike the `break` statement, `continue` does not force the termination of a loop, it merely transfers control to the next iteration.

PROGRAM CONTROL

- Consider the following continue keyword example

```
// using the continue in for structure
#include <stdio.h>

int main(void)
{
	int iNum;
	for(iNum = 1; iNum <= 10; iNum++)
	{
		// skip remaining code in loop only if iNum == 5
		if(iNum == 5)
		    continue;
		printf("%d ", iNum);
	}
	printf("\nUsed continue to skip printing the value 5\n");
	return 0;
}
```

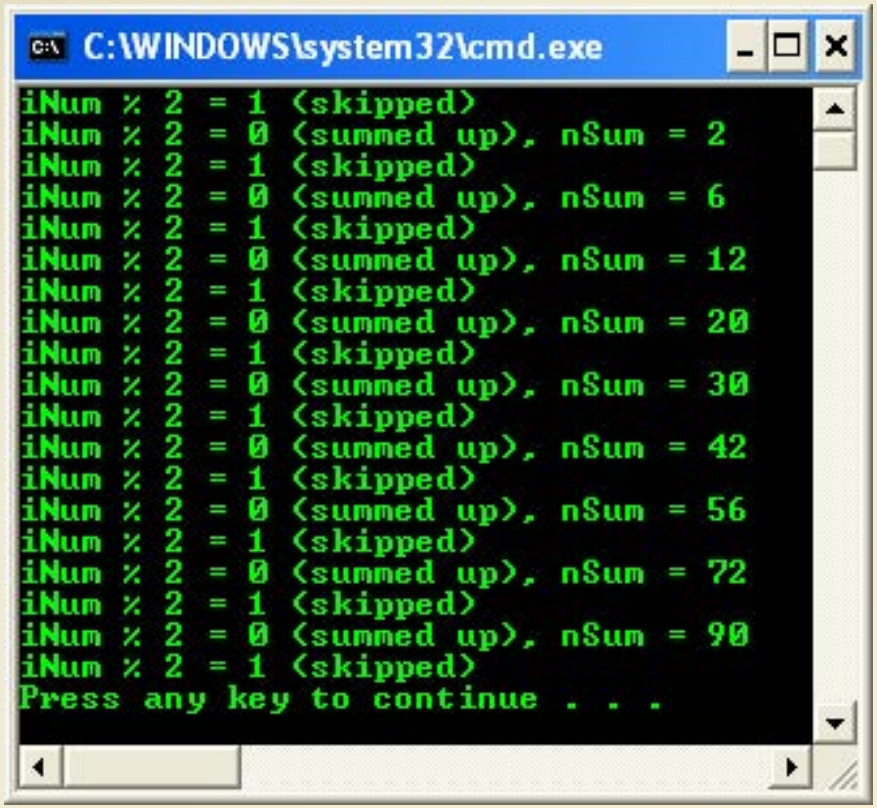


PROGRAM CONTROL

- Next consider the following continue keyword example,

```
#include <stdio.h>

int main(void)
{
        int iNum, nSum;
        for(iNum=1, nSum=0; iNum<20; iNum++)
        {
                // test value, 0 or non-zero
                if (iNum % 2)
                {
                        printf("iNum %% 2 = %d (skipped)\n", iNum % 2);
                        // executed if the test value is non-zero
                        // and repeat the for statement
                        continue;
                }
                // executed if the test value is zero and repeat the for statement
                nSum = nSum + iNum;
                printf("iNum %% 2 = %d (summed up), nSum = %d \n", iNum % 2, nSum);
        }
        return 0;
}
```

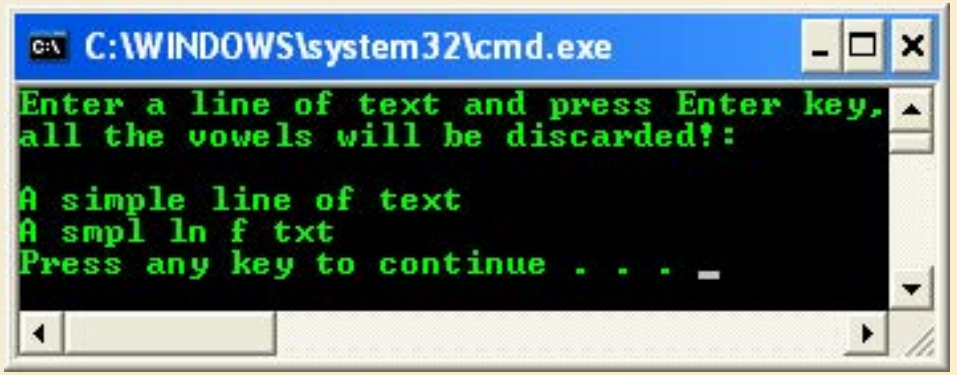
PROGRAM CONTROL



- This loop sums up the even numbers 2, 4, 6, ... and stores the value in the `nSum` variable.
- If the expression `iNum % 2` (the remainder when `iNum` is divided by 2) yields a non-zero value (i.e., if `iNum` is odd), the `continue` statement is executed and the iteration repeated (`iNum` incremented and tested).

PROGRAM CONTROL

- If it yields a zero value (i.e., if `iNum` is even), the statement `nSum = nSum + iNum;` is executed and the iteration continued.
- When a `continue` statement executes, the next iteration of the enclosing loop begins.
- The enclosing loop means the statements between the `continue` statement and the end of the loop are not executed.
- Try another continue example



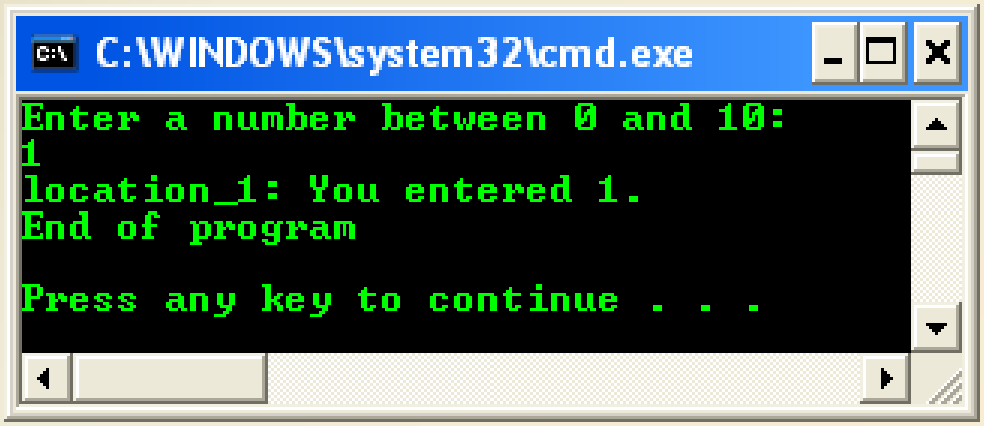
PROGRAM CONTROL

`goto` keyword

- The `goto` statement is one of C unconditional jump or branching.
- When program execution encounters a `goto` statement, execution immediately jumps, or branches, to the location specified by the `goto` statement.
- The statement is unconditional because execution always branches when a `goto` statement is came across, the branching does not depend on any condition.
- A `goto` statement and its target label must be located in the same function, although they can be in different blocks.
- Use `goto` to transfer execution both into and out of loop.
- However, using `goto` statement strongly not recommended.
- Always use other C branching statements.
- When program execution branches with a `goto` statement, no record is kept of where the execution is coming from.

PROGRAM CONTROL

- Try the following C goto keyword program example



PROGRAM CONTROL

`exit()` function

- `exit()` function normally used when a program want to terminate at any time.
- The `exit()` function terminates program execution and returns control to the Operating System.
- The syntax of the `exit()` function is,

```
exit(status);
```

Status	Description
`0` (zero)	The program terminated normally.
`1` (or non-zero)	Indicates that the program terminated with some sort of error. The return value is usually ignored.

PROGRAM CONTROL

- We must include the header file `stdlib.h` (`cstdlib` if used in C++ code).
- This header file also defines two symbolic constants for use as arguments to the `exit()` function, such as,

```
#define  EXIT_SUCCESS  0
#define  EXIT_FAILURE  1
```

- Then we can call the function like the following,

```
exit(EXIT_SUCCESS);
```

Or

```
exit(EXIT_FAILURE);
```

PROGRAM CONTROL

`atexit()` function

- Used to specify, or register, one or more functions that are automatically executed when the program terminates.
- Exit-processing function that executes prior to program termination
- These functions are executed on a last-in, first-out (LIFO) basis, the last function registered is the first function executed.
- When all functions registered by `atexit()` executed, the program terminates and returns control to the OS.
- The prototype of the `atexit()` function is located in the stdlib.h and the syntax is,

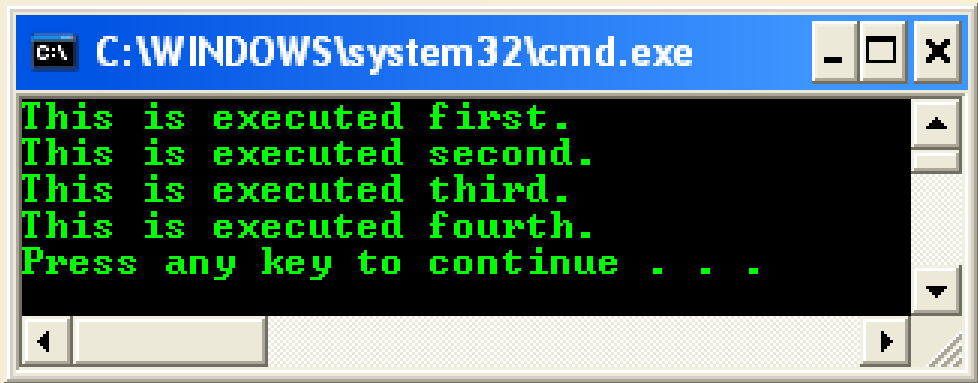
```
int  atexit(void(*funct)(void));
```

- where `funct` is the function to be called.

PROGRAM CONTROL

- `atexit()` function takes a function pointer as its argument and functions with `atexit()` must have a return type of `void`.
- The functions passed to `atexit()` cannot take parameters.
- `atexit()` uses the heap (instead of stack) to hold the registered functions.
- The following program pushes three functions onto the stack of functions to be executed when `atexit()` is called.
- When the program exits, these programs are executed on a last in, first out basis.

The atexit() function program example



PROGRAM CONTROL

`system()` function

- The `system()` function, enables the execution of OS command from the C running program.
- Can be quite useful, for example, enabling the program to do a directory listing or formatting a disk without exiting the program.
- Must include the header file stdlib.h. The syntax is,

```
system("command");
```

- The command can be either a string constant or a pointer to a string.

PROGRAM CONTROL

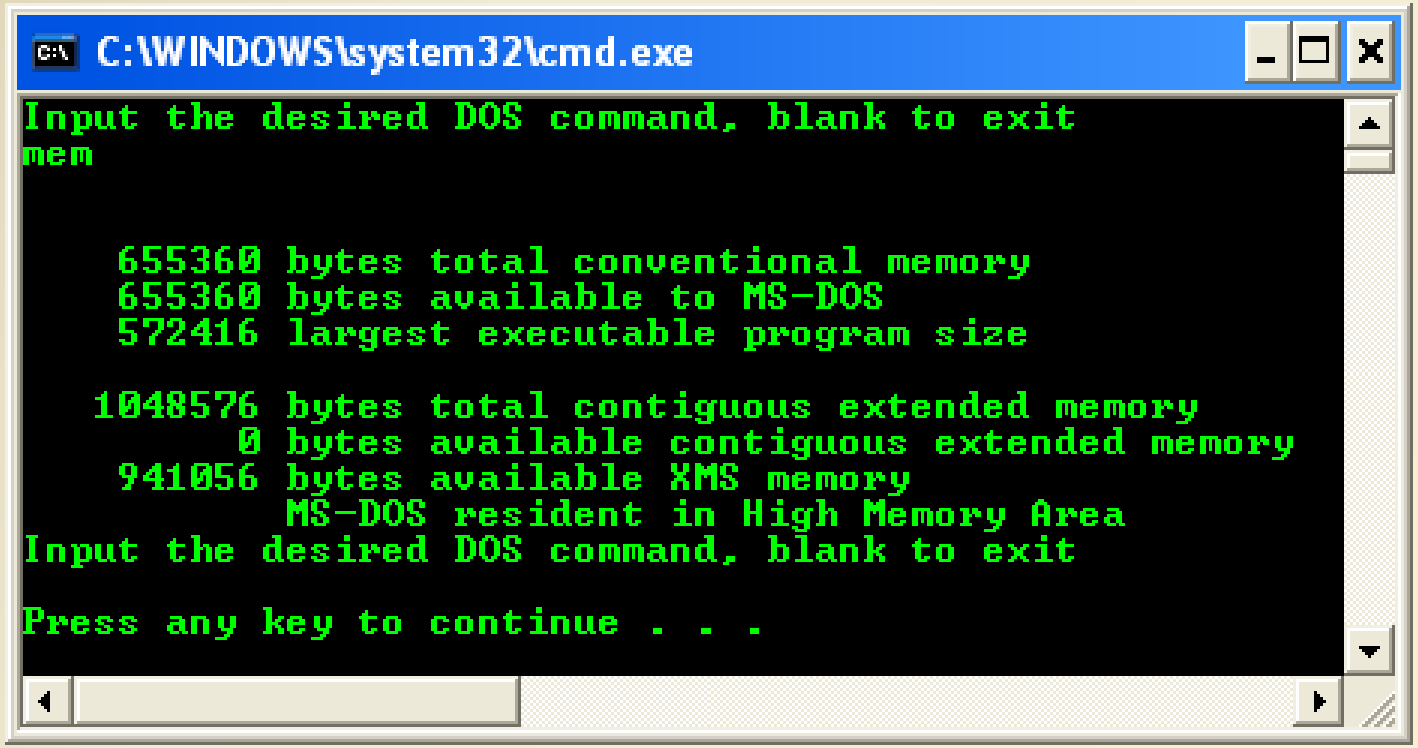
- For example, using an argument with the `system()` function,

```
char *command = "dir";
system(command);
```

- After the OS command is executed, the program continues at the location immediately following the `system()` call.
- If the command passed to the `system()` function is not a valid OS command, a bad command or file name error message is displayed before returning to the program.
- The command can also be any executable or batch file to be run.

PROGRAM CONTROL

Try the following system() program example



PROGRAM CONTROL

`return` keyword

- The `return` statement has a form,

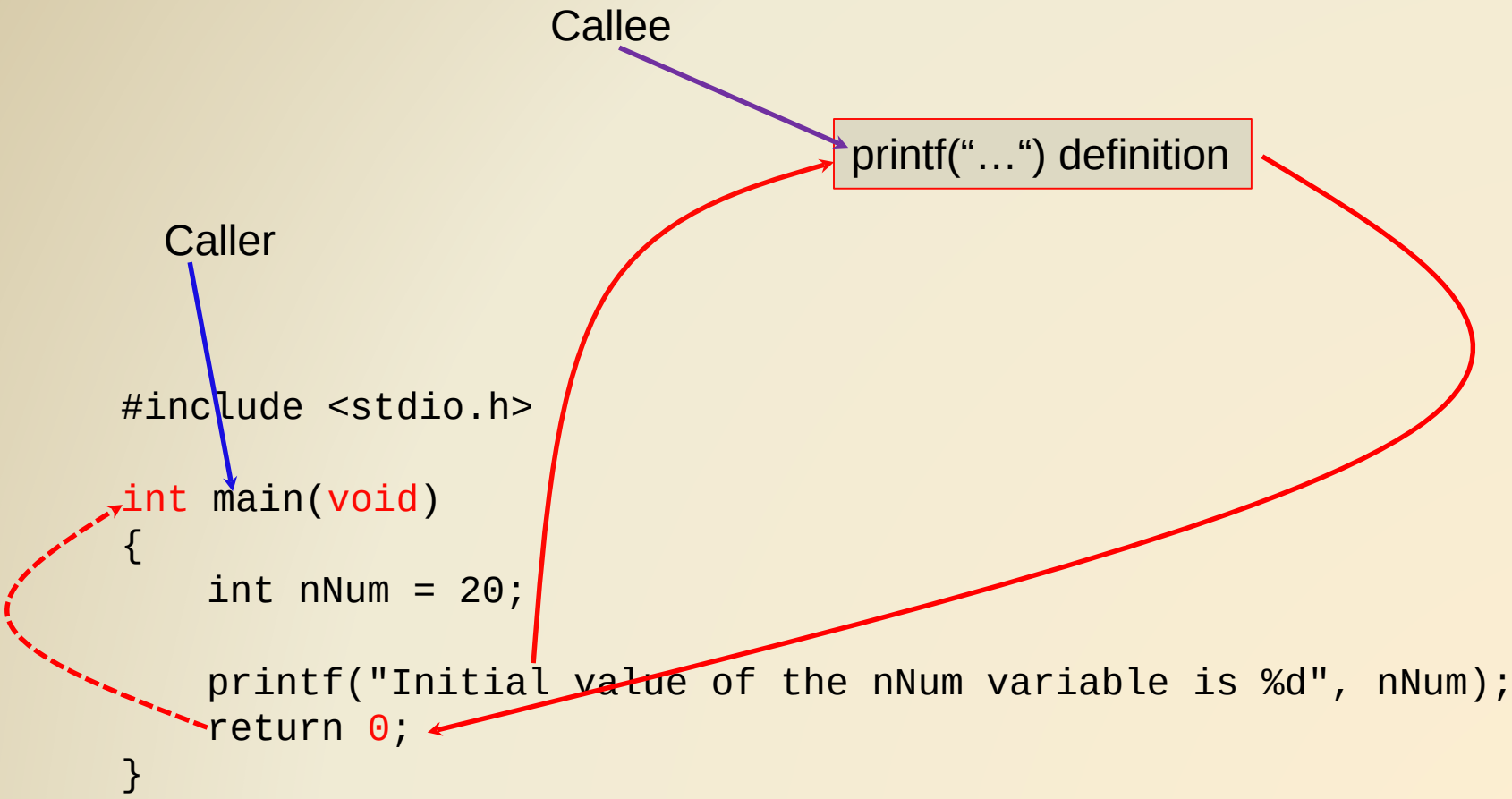
  `return` *`expression`*`;`

- The action is to terminate execution of the current function and pass the value contained in the expression (if any) to the function that invoked it.
- The value returned must be of the same type or convertible to the same type as the function's return type (type casting).
- More than one return statement may be placed in a function.
- The execution of the first `return` statement in the function automatically terminates the function.

PROGRAM CONTROL

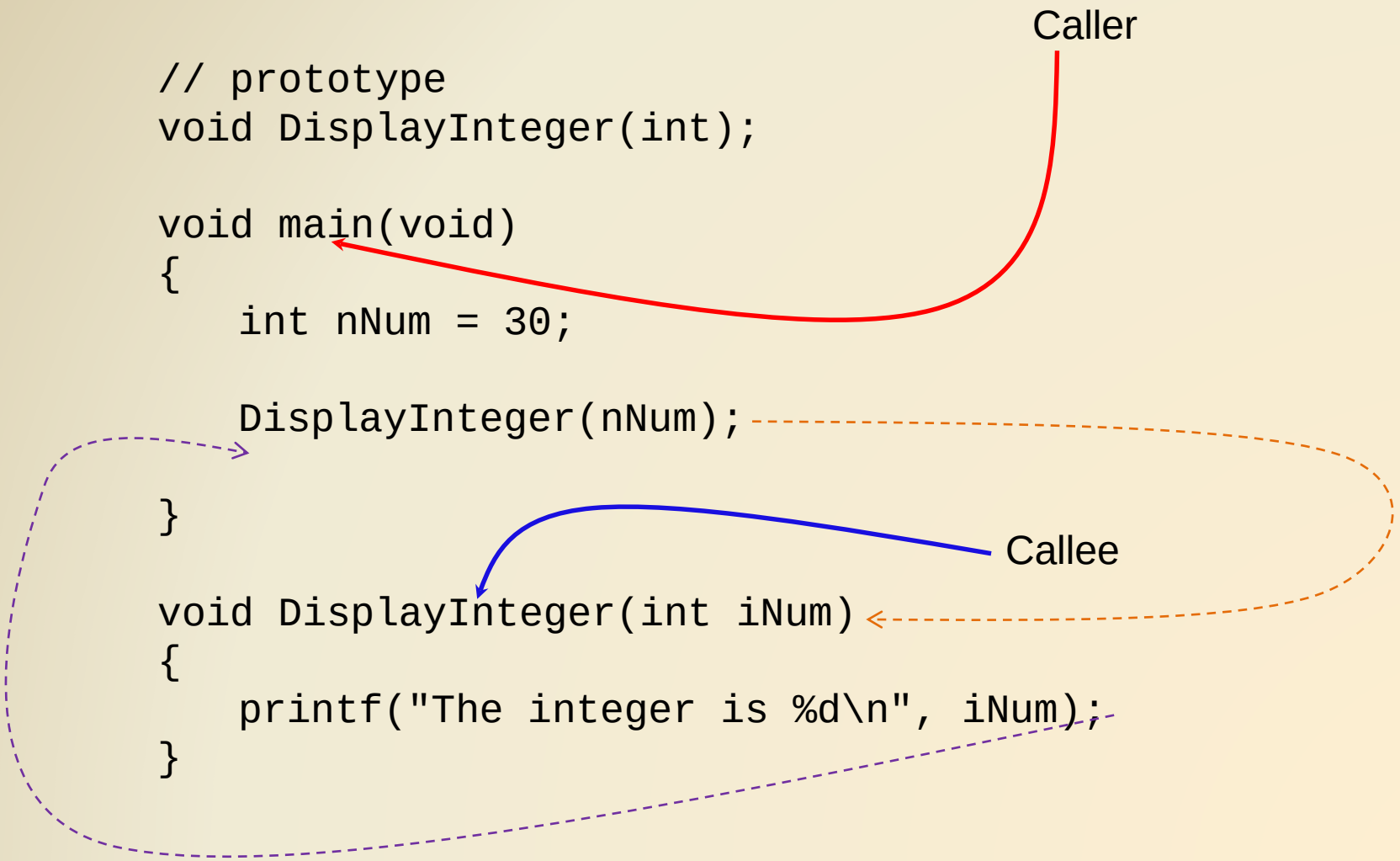
- The `main()` function has a default type `int` since it returns the value `0` (an integer) to the environment.
- A function of type `void` will not have the expression part following the keyword `return`.
- Instead, in this case, we may drop the entire `return` statement altogether.
- If a function calls another function before it is defined, then a prototype for it must be included in the calling function.
- This gives information to the compiler to look for the called function (callee).

PROGRAM CONTROL



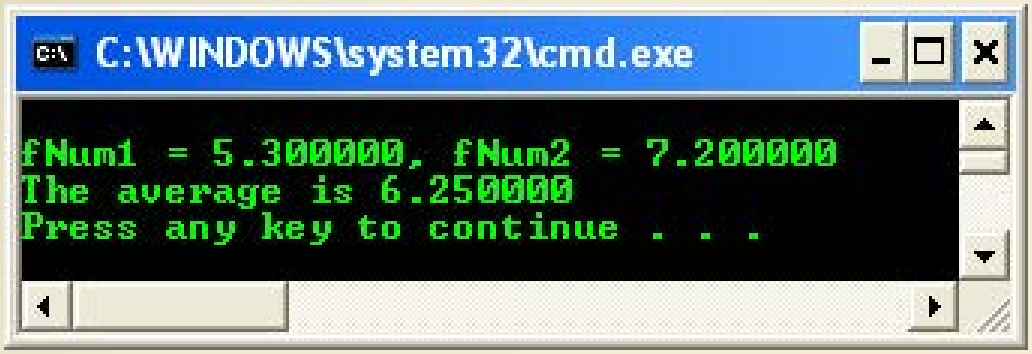
PROGRAM CONTROL

```
#include <stdio.h>

// prototype
void DisplayInteger(int);

void main(void)
{
    int nNum = 30;

    DisplayInteger(nNum);

}

void DisplayInteger(int iNum)
{
    printf("The integer is %d\n", iNum);
}
```

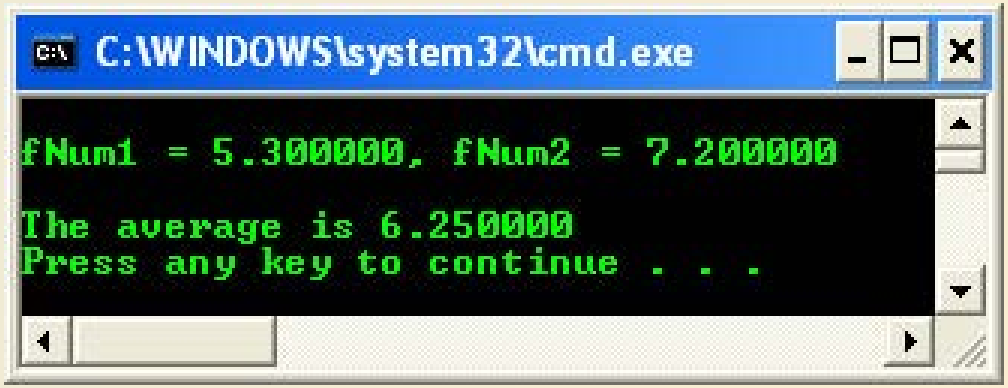


PROGRAM CONTROL

- The return keyword example 1



- The return keyword example 2



PROGRAM CONTROL

`break` keyword

Program example on using break in for loop

- Already discussed in `switch-case` constructs.
- The `break` statement terminates the execution of the nearest enclosing loop or conditional statement in which it appears. Control passes to the statement that follows the terminated statement, if any.
- Used with the conditional `switch` statement and with the `do`, `for`, and `while` loop statements.
- In a `switch` statement, `break` causes the program to execute the next statement after the `switch`. Without a `break` statement, every statement from the matched case label to the end of the `switch`, including the `default`, is executed.
- In loops, `break` terminates execution of the nearest enclosing `do`, `for`, or `while` statement. Control passes to the statement that follows the terminated statement, if any.
- Within nested statements, the `break` statement terminates only the `do`, `for`, `switch`, or `while` statement that immediately encloses it. You can use a `return` or `goto` statement to transfer control from within more deeply nested structures.

PROGRAM CONTROL

`abort()` and `terminate()` functions

Function	Description
`abort()`	Abort current process and returns error code defined in `stdlib.h`
`terminate()`	Used when a handler for an exception cannot be found. The default action to terminate is to call `abort()` and causes immediate program termination. It is defined in `except.h` (Microsoft uses eh.h and only compiled in C++).

1. The syntax is,

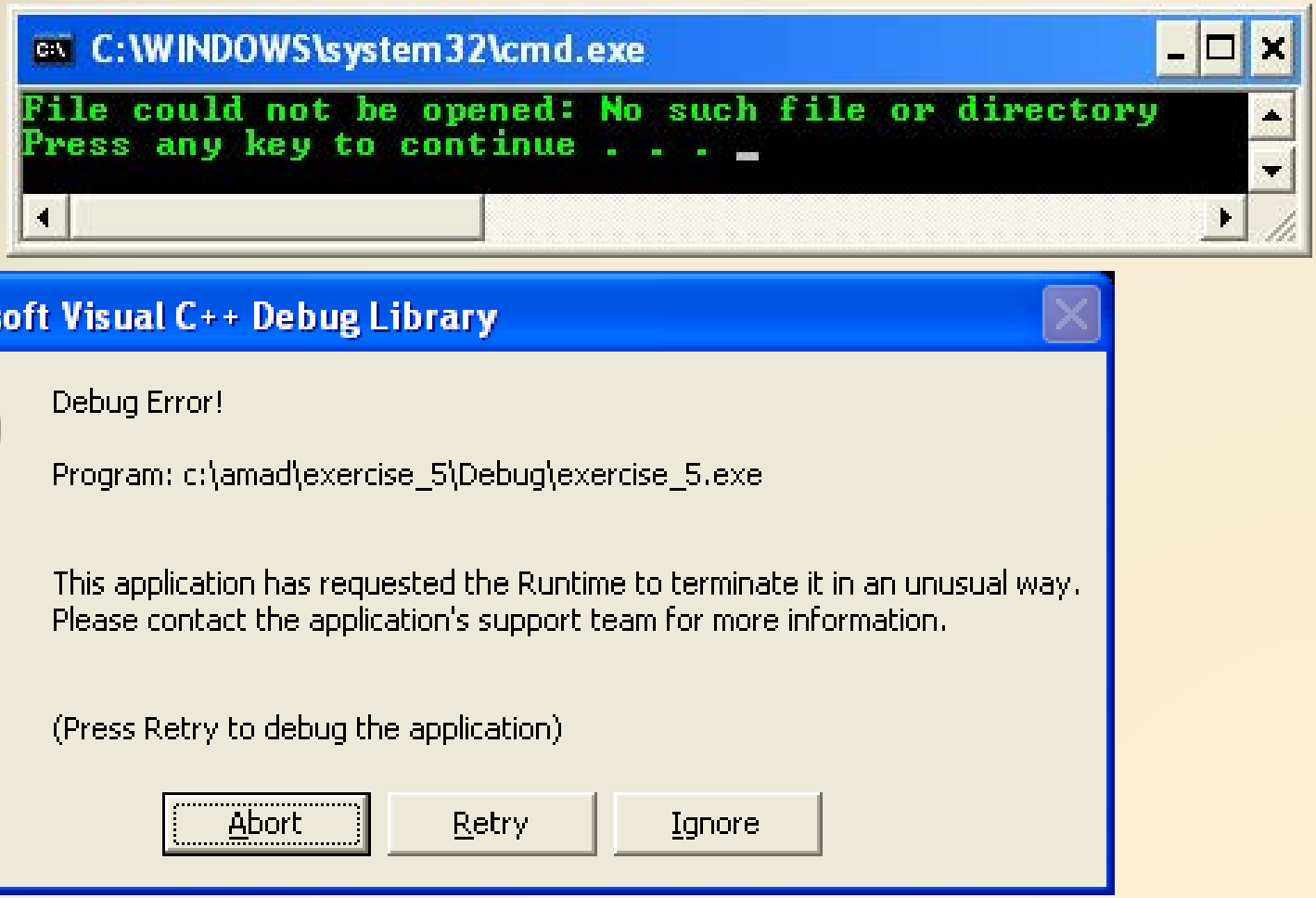
```
void abort( void );
```

1. `abort()` does not return control to the calling process. By default, it terminates the current process and returns an exit code of 3.
2. By default, the abort routine prints the message:

```
"This application has requested the Runtime to terminate it in an
unusual way. Please contact the application's support team for
more information."
```

PROGRAM CONTROL

- The abort() program example



PROGRAM CONTROL

- `terminate()` function calls `abort()` or a function you specify using `set_terminate()`.
- `terminate()` function is used with C++ exception handling and is called in the following cases:
  1. A matching catch handler cannot be found for a thrown C++ exception.
  2. An exception is thrown by a destructor function during stack unwind.
  3. The stack is corrupted after throwing an exception.
- `terminate()` calls `abort()` by default. You can change this default by writing your own termination function and calling `set_terminate()` with the name of your function as its argument.
- `terminate()` calls the last function given as an argument to `set_terminate()`.

PROGRAM CONTROL

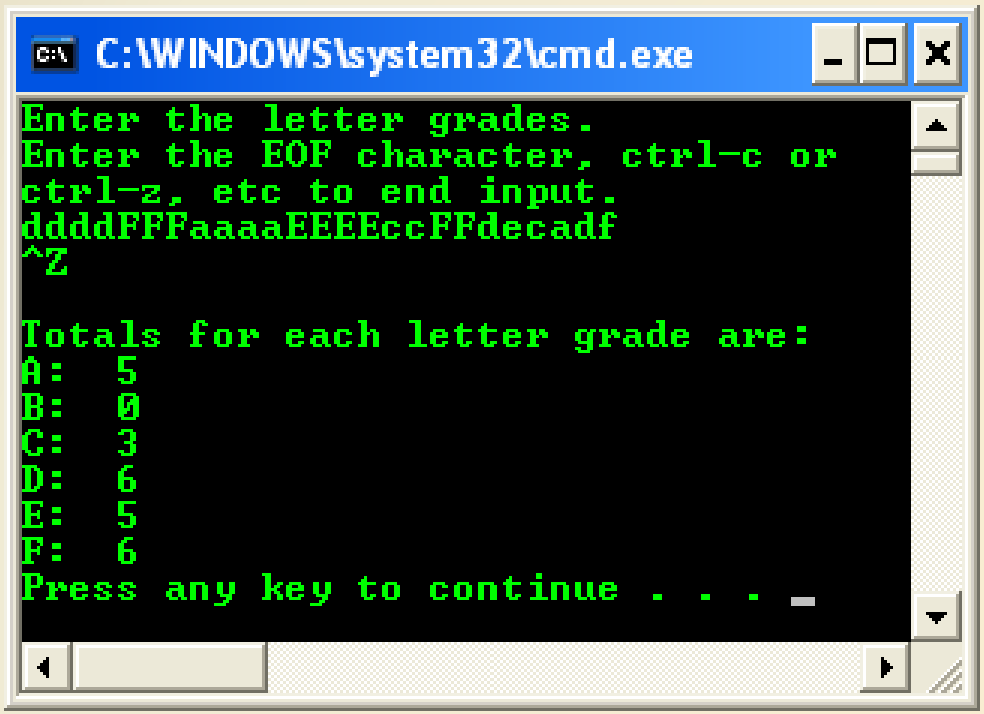
- The terminate() program example (C++)

PROGRAM CONTROL
EOF

- We use EOF (acronym, stands for End Of File), normally has the value –1, as the sentinel value.
- The user types a system-dependent keystroke combination to mean end of file that means 'I have no more data to enter'.
- EOF is a symbolic integer constant defined in the <stdio.h> header file.
- The keystroke combinations for entering EOF are system dependent.
- On UNIX systems and many others, the EOF is <Return key> or ctrl-z or ctrl-d.
- On other system such as old DEC VAX VMS or Microsoft Corp MS-DOS, the EOF is ctrl-z.

PROGRAM CONTROL

- The EOF program example. (Press <Enter> + <Ctrl+z> for EOF)
- If the value assigned to grade is equal to EOF, the program terminates.



End of C Program Controls + Flow Charts