CONTROL OF FLOW OF C PROGRAMMING - III

Nested if constructs

When the expression evaluated by an `if` statement yields a TRUE, then the statement following the `if` statement is executed. This can be another `if` statement. In nested `if` statements every `else` clause is associated with the last `if` statement.

else if statement

When we have test a single variable for three different values we can use the simpler else `if` `else if` statement instead of three separate `if` statements. General format of this statement is:

```c
if (expression1)
{
    statements;
}
else if (expression2)
{
    statements;
}
else
{
    statements;
}
```
Program 5.7

```c
/* else if construct demonstration*/
#include <stdio.h>
main()
{
    long cost;

    printf("Enter the turnover of your company in dollars:\n");
    scanf("%ld", &revenue);
    printf("Enter the expenses of your company in dollars :\n");
    scanf("%ld", &cost);
    if(revenue > cost)
        printf("Profit for this year is $%ld\n", revenue - cost);
    else if(cost > revenue)
        printf("Loss for this year is $%ld\n", cost - revenue);
    else
        printf("Revenue equals cost for this year ! \n");
}
```

The switch statement

It is commonly seen in applications that the value of a variable is successively compared against different values. It becomes cumbersome to write a number of if and else if statements and readability of a program reduces. A more elegant way to handle this is by using the switch statement. The format of this is as follows:

```c
switch (expression1)
{
    case val1 :
```
program statements;
break;
case val2 :
    program statements;
    break;
case val3 :
    program statements;
    break;
case val4 :
    program statements;
    break;
case val5 :
    program statements;
    break;
default :
    program statements;
    break;
}

The expression1 is repeatedly compared against values val1, val2 and so on till a match is found. Then the corresponding program statements are executed. Every set of statements needs a break statement otherwise the program execution will continue into the next case statement that satisfies the expression. There is a special default statement at the end which gets executed when the value of expression1 does not match any of the case values.

/* Example program for Switch case statement */
#include <stdio.h>
main()
{
    float book_price, net_price;
    float discount;
    int cust_code;
    printf("Enter the price of the book : \n");
scanf("%f", &book_price);
printf("Enter the customer code :\n");
scanf("%d", &cust_code);

switch (code)
{
  case 1 :              /* Registered customers */
    discount = 0.1;
    net_price = book_price - (book_price * discount);
    break;

  case 2 :              /* Wholesale dealers */
    discount = 0.15;
    net_price = book_price - (book_price * discount);
    break;

  case 3 :              /* Internal employees */
    discount = 0.17;
    net_price = book_price - (book_price * discount);
    break;

  default :              /* First time customers */
    discount = 0.05;
    net_price = book_price - (book_price * discount);
    break;
}
printf("Net price of the book is %f\n", net_price);
goto statement

C language includes the less used goto statement. Same as the break and continue statements, generally it is recommended to avoid using this statement. But one has to decide for oneself whether a program needs it or not and use it judiciously. > A goto statement causes a branch to be made to a specified point in a program. This point is denoted by a label which is a name followed by a colon. A label is formed with the same rules as variable names. This can be located anywhere in a program, either before or after the goto statement.

Example:
go to end_of_loop;

program statements

end_of_loop: printf("End of loop\n");

Source: http://www.peoi.org/Courses/Coursesen/cprog/frame5.html