Decision making statements in C

The `if ()` statement

The `if` keyword in C programming language is used whenever we want to make decisions in the program.

Here's the general syntax.

```c
#include<stdio.h>
int main()
{
    if (/* condition goes here */)
    {
        /* if the condition is non-zero (true), this code will execute */
    }
    return 0;
}
```

Relational and Equivalence Expressions:

`a < b` 1 if a is less than b, 0 otherwise. `a > b` 1 if a is greater than b, 0 otherwise. `a <= b` 1 if a is less than or equal to b, 0 otherwise. `a >= b` 1 if a is greater than or equal to b, 0 otherwise. `a == b` 1 if a is equal to b, 0 otherwise. `a != b` 1 if a is not equal to b, 0 otherwise

Logical Expressions

`a || b` when EITHER a or b is true (or both), the result is 1, otherwise the result is 0. `a && b` when BOTH a and b are true, the result is 1, otherwise the result is 0. `!a` when a is true, the result is 0, when a is 0, the result is 1.

Here's an example of a larger logical expression. In the statement:

`e = ((a && b) || (c > d));`

e is set equal to 1 if a and b are non-zero, or if c is greater than d. In all other cases, e is set to 0. C uses short circuit evaluation of logical expressions. That is to say, once it is able to determine the truth of a logical expression, it does no further evaluation. This is often useful as in the following:

`int i, m; .... if (i < 12 && m > 3)`

`{`

`// do something here`

`}`

Bitwise Boolean Expressions

The bitwise operators work bit by bit on the operands. The operands must be of integral type (one of the types used for integers). The six bitwise operators are & (AND), | (OR), ^ (exclusive OR, commonly called XOR), ~ (NOT, which changes 1 to 0 and 0 to 1), << (shift left), and >> (shift right). The negation operator is a unary operator which precedes the operand. The others are binary operators which lie between the two operands. The precedence of these operators is lower than that of the relational and equivalence operators; it is often required to parenthesize expressions involving bitwise operators. For this section, recall that a number starting with 0x is hexadecimal, or hex for short. Unlike the normal decimal system using powers of 10 and digits 0123456789, hex uses powers of 16 and digits 0123456789abcdef. Hexadecimal is commonly used in C programs because a programmer can quickly convert it to or from binary (powers of 2 and digits 01). C does not directly support binary notation, which would be really verbose anyway. `a & b` bitwise boolean and of a and b 0xc & 0xa produces the value 0x8 (in binary, 1100 & 1010 produces 1000) `a | b` bitwise boolean or of a and b 0xc | 0xa produces the value 0xe (in binary, 1100 | 1010 produces 1110) `a ^ b` bitwise xor of a and b 0xc ^ 0xa produces the value 0x6 (in binary, 1100 ^ 1010 produces 0110) `~a` bitwise complement of a. ~0xc produces the value -1-0xc (in binary, ~1100 produces ...11110011 where "..." may be many more 1 bits) `a << b` shift a left by b (multiply a by 2b) 0xc << 1 produces the value 0x18 (in binary, 1100 << 1 produces the value 11000) `a >> b` shift a right by b (divide a by 2b) 0xc >> 1 produces the value 0x6 (in binary, 1100 >> 1 produces the value 110)

The `if () - else` statement

If-Else provides a way to instruct the computer to execute a block of code only if certain conditions have been met.

The general syntax of an If-Else construct is:

```
if (/* condition goes here */)
{
    /* if the condition is non-zero (true), this code will execute */
}
else
{
    /* if the condition is 0 (false), this code will execute */
}
```

The `if () - else if()` statement

General Syntax

```
if (/* condition goes here */)
{
    /* if the condition is non-zero (true), this code will execute */
}
else if (/* condition goes here */)
{
    /* if the condition is non-zero (true), this code will execute */
}
else
{
    /* if the condition is 0 (false), this code will execute */
}
```

Nested `if()` s

You can also use multiple `if()` statements inside each other.

For example,

```
#include <stdio.h>
int main ()
{
   int a = 100;
   int b = 200;
   /* check the condition */
   if( a == 100 )
   {  /* if condition is true then check the following */
      if( b == 200 )
      {  /* if condition is true then print the following */
         printf("Value of a is 100 and b is 200\n" );
      }
   }
   printf("Exact value of a is : %d\n", a );
   printf("Exact value of b is : %d\n", b );
   return 0;
}
```

The conditional expression

Also called the ternary operator, is the one and only operator in C programming language that operates on three operands.

You might have observed something like `( ) ? : ;`

General syntax is

```
(/*test expression*/) ? /* if the condition is non-zero (true), this code will execute
*/ : /* if the condition is non-zero (true), this code will execute */ ;
```

important : note that the above code is written in a single line!

```
#include<stdio.h>
int main()
{
    (1) ? printf("true") : printf("false") ;
    // note that the test expression is always true here and hence the output is "true"
    return 0;
}
```

Switch Case

Say you write a program where the user inputs a number 1-5 (corresponding to student grades, A(represented as 1)-D(4) and F(5)), stores it in a variable grade and the program responds by printing to the screen the associated letter grade. If you implemented this using If-Else, your code would look something like this:

```
int grade;
if (grade == 1)
{
    printf("A\n");
}
else if (grade == 2)
{
    printf("B\n");
}
else if /* etc. etc. */
```

You can use the same logic with a switch case!

The general syntax of the switch case construct is

```
switch (/* variable goes here */)
{
    case /* potential value of the variable*/:
    /* code */
    case /* a different potential value */:
    /* different code */
    /* insert additional cases as needed */
    default:
    /* more code */
}
```

Then, your code would look something like

```
int grade;
switch (grade)
{
    case 1  :   printf("A\n");  break;
    case 2  :   printf("B\n");  break;
        // ... and other cases
        default :   printf("wrong choice");
}
```

Very important: Typically, the last statement for each case is a break statement. This causes program execution to jump to the statement following the closing bracket of the switch statement, which is what one would normally want to happen. However if the break statement is omitted, program execution continues with the first line of the next case, if any. This is called a fall-through. When a programmer desires this action, a comment should be placed at the end of the block of statements indicating the desire to fall through. Otherwise another programmer maintaining the code could consider the omission of the 'break' to be an error, and unknowingly 'correct' the problem.

```
switch (someVariable)
{
    case 1:
    printf("This code handles case 1\n");
    break;
    case 2:
    printf("This prints when someVariable is 2, along with...\n");
    /* FALL THROUGH */
    case 3:
    printf("This prints when someVariable is either 2 or 3.\n" );
    break;
}
```