3. Operators and control statements

3.1. Introduction

C provides various operators e.g. algebraic and boolean operators etc., which can be used to perform different mathematical operations. Further, various control structure of C can be used to perform these tasks repetitively or under suitable conditions. In this chapter, such operators and control structures are used to perform operations on various data-types.

3.2. Operators

Operators are the symbols which are used to perform certain operations on the data e.g. addition, subtraction and comparison etc. There are various types of operators in C, which are shown in this section; also, most of these operators are used with decision statements, therefore usage of these operators are shown in Section 3.3.

3.2.1. Arithmetic operators

Table 3.1 shows the list of arithmetic operators in C. These operations are used to perform various mathematical operations on 'integer', 'float' and 'double' data types. Functions of all the operators are straightforward except for '++' and '–', which are used to increment or decrement the value of the variable by '1' respectively. These operators can be used as 'i++ or i–' and '++i or –i'. The '++i' operation indicates that 'increase the value of by '1' and then use that value; whereas 'i++' indicates that 'use the value of i first' and then increment it by 1. These two operations are shown in Listing 3.1.

Table 3.1 Arithmetic Operators

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus (remainder) (e.g. 3%2 = 1)
++	"Increment (e.g. if a = 1, then a++ = 2)"
–	"Decrement (e.g. if a = 1, then a– = 0)"
+= -= *= /=	short notation (e.g.'a += 3' indicates 'a = a+3'

Explanation Listing 3.1

In the listing, the variable 'i' is initialized as '1' at Line 7, and printed at Line 9. Next, 'i++' operation is printed at Line 12; note that, first '1' is printed by this line; and then the value is increased. This increase in value is verified by print 'i' again at Line 13, where the result is 2. Next, '++i' operation is used at Line 16, where the value is increase first and then printed, therefore the result is '3'. Finally, to check the value stored in 'i', it is printed again at Line 17.

Listing 3.1 Difference in 'i++' and '++i'

```
1  // incrementEx.c
2
3  #include <stdio.h>
4
5  int main(void)
6  {
7      int i = 1;
8
9      printf("i = %d\n\n", i);  // 1
10
11     // print first, then increment
12     printf("i++ = %d (i.e. access first and then increment): \n", i++); // 1
13     printf("i = %d\n\n", i); // 2 (i.e. i is incremented by above statement)
14
15     // increment first, then print
16     printf("++i = %d (i.e. increment first and then access):\n", ++i); // 3
17     printf("i = %d\n", i); // 3
18
19     return 0;
20 }
21
22 /* Outputs
23 i = 1
24
25 i++ = 1 (i.e. access first and then increment):
26 i = 2
27
28 ++i = 3 (i.e. increment first and then access):
29 i = 3
30 */
```

3.2.2. Relational operators

Relations operators are used of checking various equality conditions e.g 'greater than', 'equal' or 'less than' etc. These operators are shown in Table 3.2.

Table 3.2 Relational Operators

Condition	Symbol
equal	==
not equal	!=
greater	>
smaller	<
greater or equal	>=
smaller or equal	<=

3.2.3. Bitwise operators

Bitwise operators are used with boolean data type to calculate the results of boolean expressions or to perform shift operations on boolean data. These operators are shown in Table 3.3. Some of these operations are shown in Listing 3.2.

Table 3.3 Bitwise operators

Operators	Symbol
OR	|
AND	&
XOR	^
NOT	~
Left Shift	<<
Right Shift	>>

Explanation Listing 3.2

Most of the operations in the listing are straightforward i.e. right shift(Line 12), left shift (Line 13) and xor (Line 17). **But, look at the 'not operation' carefully, it may generate errors in the result in NIOS design**. Since, type 'int' is 32 bit long, therefore, a = 2 is saved as '0000-0002' and when 'not' operation is performed on it, then result is 'ffff-fffd' (not '0000-000d'), at Lines 20-21; therefore, we need to perform '& (and)' operation with the result, to suppress all the 'f' in the result (Line 24).

Listing 3.2 Bitwise operator

```c
// shiftOpEx.c

#include <stdio.h>

int main(){
    int a = 0x2;  // 0010

    int b, c;
    int xor_op, not_op_bad, not_op_good;

    // shift right by one bit and save to b
    b = a >> 1;  // 0001 = 1
    c = a << 2;  // 1000 = 8

    printf("a = %x, b = %x, c = %x\n", a, b, c);

    xor_op = a ^ b; // 0011 = 3
    printf("xor_op = %x\n", xor_op);

    not_op_bad = ~a; // fffffffd (expected output = 1101 = 13 = 'd')
    printf("not_op_bad = %x\n", not_op_bad);

    // suppress all 'f' (i.e. make them 0) by 'and-operation' of 'a' with '0000000f' = 15
    not_op_good = ~a & 0xf; // or not_op_good = ~a & 15;
    printf("not_op_good = %x\n", not_op_good);
}

/* Outputs
a = 2, b = 1, c = 8
xor_op = 3
not_op_bad = fffffffd
not_op_good = d
*/
```

3.2.4. Logical operators

Logical operators are used to perform logic ‘and’, ‘or’ and ‘not’ operations, which are shown in Table 3.4. Unlike, ‘bitwise logic operators’ these operators do not check the second operator if the first operator gives the result e.g. in ‘0 && 1 = 0’, the result does not depend on the value of second operand i.e. ‘1’; because first operand is ‘0’, therefore the result will be zero, whether the second operand is ‘0’ or ‘1’.

Table 3.4 Logical operators

Operator	Description
&&	Logical AND operator
||	Logical OR Operator
!	Logical NOT Operator

3.3. Decision statements

Control structure can be devided into two categories i.e. decision statements and loops. In this section, simple C codes are presented to explain decision statements available in the language; whereas loops are discussed in next section.

3.3.1. If-else statements

If-else statements are used to define different actions for different conditions. Symbols for such conditions are given in Table 3.2, which can be used as shown in Listing 3.3. Three examples are shown in this section for three types of statements i.e. **if**, **if-else** and **if- 'else if' -else**.

Explanation Listing 3.3

Listing 3.3 checks whether the number stored in variable 'x' is even or not.

In the listing, variable 'x' of type 'int' is defined at Line 8; also, this variable is initialize with value 3. In Lines 11 and 17, the '%' sign calculates the remainder of division x/2 and checks whether remainder is zero or not respectively. Then, Lines 12-13 will be printed if remainder is zero, else Line 18-19 will be printed.

Notice that, { } are used with if-statements e.g. at Lines 11 and 14. These brackets tell the compiler that, all the statements within them must be executed only when the condition in corresponding 'if' statement is satisfied i.e. 12-13 will be executed, when Line 11 is true. Further, if we remove the brackets from these line, then Line 12 will depend on the 'if' statement, whereas Line 13 will be printed all the time.

Listing 3.3 If statement

```
1   // ifEx.c
2   // checks whether number is even or odd...
3
4   #include <stdio.h>
5
6   int main(void)
7   {
8       int x=3;
9
10      // check if number is even
11      if(x % 2 == 0){
12          printf("Number is even.");
13          printf("Thank you...");
14      }
15
16      // check if number is odd
17      if(x % 2 != 0){
18          printf("Number is odd.");
19          printf("Thank you...");
20      }
21
22      return 0;
23  }
24
25  /* Outputs
26  Number is odd.
27  Thank you...
28  */
```

Explanation Listing 3.4

As we know that a number can not be even and odd at the same time. In such cases, we can use if-else statement. Listing 3.3 can be rewritten using 'if-else' statement as shown in Listing 3.4.

Here, only one condition is specified i.e. at Line 10. Compiler will check this line, if this is true, then Line 11-12 will be printed, otherwise compiler will go to 'else' statement at Line 14 and Lines 15-16 will be printed.

Listing 3.4 If-else statement

```
1   //ifElseEx.c
2   // checks whether number is even or odd...
3
4   #include <stdio.h>
5
6   int main(void)
7   {
8       int x=3;
9
10      if (x % 2 == 0){
11          printf("Number is even.");
12          printf("Thank you...");
13      }
14      else{
15          printf("Number is odd.");
16          printf("Thank you...");
17      }
18
19      return 0;
20  }
21
22  /* Outputs
23  Number is odd.
24  Thank you...
25  */
```

- In previous case, there were only two conditions which were implemented using 'if-else statement'. If there are more than two conditions then 'else if' can be used as shown in this example. Further, 'if else-if else' block can contain any number of 'else-if' statement between one 'if' and one 'else' statement.

Note

Note that, the operator 'or' at Line 18 of Listing 3.5, will generate error, if we execute the code using 'gcc ElseIFEx.c -o out', as this operator for C++ compiler only. This code can be executed with command 'g++ ElseIFEx.c -o out'.

Explanation Listing 3.5

In this listing, Line 11 get the value of variable 'grade' from the user. The possible value of grades are A, a, B, b, C, c, D and d. If grade is 'A or a', then Line 15 will be printed. Note that, the 'or' operation is used in three ways, i.e. '||', '|' and 'or' at Lines 14, 16 and 18 respectively. If grade value provided by the user is 'B or b', then compiler will first check the Line 14, which is false, therefore it will go to next 'else-if' statement at Line 16, since it is true, therefore Line 17 will be printed; and compiler will leave the if-block. If we provide the wrong grade, e.g. Z, then compiler will check all the statements i.e. Lines 14, 16, 18 and 20; and finally reach to 'else' block and print the Line 23.

Listing 3.5 'If else-if else' statement

```c
// ElseIfEx.c
// print message based on grades...

#include <stdio.h>

int main(void)
{
    char grade;

    printf("Enter the grade - A, B, C or D: \t");
    scanf(" %c", &grade);  // get the value from user

    // if grade is A or a
    if (grade == 'A' || grade == 'a')   // logical operator
        printf("Excellent Student");
    else if ((grade == 'B') | (grade == 'b'))   // bitwise operator
        printf("Very Good Student");
    else if (grade == 'C' or grade == 'c')  // logical 'or' operator
        printf("Good Student");
    else if (grade == 'D' or grade == 'd')   // or is C++ feature (not C)
        printf("Need improvements");
    else
        printf("Invalid grade");

    return 0;
}

/* Outputs
Enter the grade - A, B, C or D:     b
Very Good Student
*/
```

Explanation Listing 3.6

We can use various if statements inside the other if statements, which are known as nested statements. Listing 3.6 is the example of **nested loop**, where divisibility of the number with 2 and 3 is checked.

The listing checks whether the number is divisible by 2 and 3 using nested if-else statements. Nested statements are used for 'if statement' at Line 10. Here, line 10 checks whether the number is division by 2 or not; if divisible, then compiler goes to nested loop (at line 11) and checks whether number is divisible by 3 or not; if divisible, then Line 12 will be printed, else Line 15 and 16 will be printed. Note that, line 16 prints the value of remainder when number is divided by 3. Next, if number is not divisible by 2 (at Line 10), then compiler will directly go to the Line 19; and if number is divisible by 3, then Line 20 will be printed. If number is not divisible by 3 as well, then compiler will go to Line 22 (i.e. else statement) and finally Lines 23-25 will be printed.

Listing 3.6 'Nested If else-if else' statement

```c
// ifElifElse.c
// Nested if-else statement to check the divisibility with 2 and 3.

#include <stdio.h>

int main(void)
{
    int x=4;

    if (x % 2 == 0){
        if(x % 3 == 0){
            printf("Number is divisible by 2 and 3.");
        }
        else{
            printf("Number is divisible by 2 only\n");
            printf("x%%3 = %d\n", x%3);  // note : two %% are used to print '%'
        }
    }
    else if (x % 3 == 0){
        printf("Number is divisible by 3 only.");
    }
    else{
        printf("Number is not divisible by 2 and 3\n");
        printf("x%%2 = %d\n", x%2);
        printf("x%%3 = %d\n", x%3);
    }

    return 0;
}

/* Outputs
Number is divisible by 2 only
x%3 = 1
*/
```

3.3.2. Switch-case-default statements

Similar to 'if statements', the switch-case statements can be used to perform certain operations, only if the specific conditions are met, as shown in Listing 3.7.

Explanation Listing 3.7

Operation of this listing is same as the Listing 3.5, but the condition is only checked for uppercase letter. We can add more case-statements for considering the lowercase letters as well. Here, the switch-statement depends on variable 'grade' (Line 10); and it is initialized with value 'B' at Line 8; and Lines 10-22 are the switch-statement, which contain various 'case-statements' (i.e. Lines 11, 14 and 17) and one 'default' statement at Line 20.

When we execute this code, the compiler will reach to 'switch statement (Line 10)' and then first go to Line 11 and compare the value of grade i.e. 'B' with 'A', since these are not equal therefore it will go to next 'case statement' at Line 15. Now, the condition is matched, therefore Line 15 will be printed out and then Line 16 will break the execution of switch statement, i.e. the complier will exit from the switch statement and will reach to line 23. '**Note that, unlike if-else-statements, the compiler does not automatically quit the switch-statement, a separate 'break' statement is required to exit the case. If we do not use 'break' statement, then compiler will check all the 'cases' and finally execute the 'default' block. More specifically, we use the 'break' statement to avoid the execution of 'default' block.**

Listing 3.7 Switch-case statement

```c
//switchCase.c
// print message based on grade-value

#include <stdio.h>

int main(void)
{
    char grade = 'B';

    switch(grade){
        case ('A'):
            printf("A: performance is excellent");
            break;
        case 'B' :
            printf("B: performance is good");
            break;
        case 'C' :
            printf("C: performance is not bad");
            break;
        default :
            printf("Please, work hard...");
    }

    return 0;
}

/* Outputs
B: performance is good
*/
```

3.3.3. Conditional operator (? :)

The conditional operator can be used as 'if-else' statements as shown Listing 3.8 and Listing 3.9. In this section, 'const char *' is used to define character-array, which is discussed in Section 2.7.

Explanation Listing 3.8

In conditional operator, the condition is specified before '?' and the 'true and false' results are seperated by ':'. For example, in this listing, condition '(a %2)== 0)' is specified before '?' at Line 14. Note that, the result before ':' (i.e. even) is assigned to variable 'result' if the condition is true, otherwise the result after ':' (i.e. odd) will be assigned to variable 'result'. Finally, Line 16 will display the message for even or odd number.

Listing 3.8 Condition operator as 'if-else' statement

```c
// conditionalOperatorEx.c
// check even and odd number

#include <stdio.h>

int main(void)
{
    int a;
    const char *result;  // character array

    printf("Enter a number:  ");
    scanf("%d", &a);

    result = (a%2 == 0 ) ? "even" : "odd";

    printf("number is %s", result);

    return 0;
}

/* Outputs
Enter a number:  4
number is even
*/
```

Explanation Listing 3.9

Conditional operator can be used as 'if- else if- else' statement as well. For this, instead of assigning the value for 'false condition' after ':', we should put another condition-checking statement; e.g. in the listing, Line 14 ends with ':' and then at Line 15 starts with another condition. Now, if the condition at Line 14 (i.e. a % == 0) does not satisfy, then condition at Line 15 will be checked. Lastly, if none of the condition is matched, then Line 17 will be printed.

Listing 3.9 Condition operator as 'if-else if-else' statement

```c
// conditionalOperatorEx2.c

#include <stdio.h>

int main(void)
{
    int a;

    const char *result;

    printf("Enter a number:  ");
    scanf("%d", &a);

    result = (a % 2 == 0 ) ? "number is divisible by 2" :
             (a % 3 == 0) ? "number is divisible by 3" :
             (a % 5 == 0) ? "Number is divisible by 5" :
             "number is not divisible with 2, 3 and 5";

    printf(result);

    return 0;
}

/* Outputs
Enter a number:  9
number is divisible by 3

Enter a number:  29
number is not divisible with 2, 3 and 5
*/
```

3.4. Loops

Loops are used to perform the operations repetitively. Three types of loop conditions are discussed in this section i.e. ‘for loop’, ‘while loop’ and ‘do while loop’.

3.4.1. For loop

Note

Note that, ‘int j’ inside the ‘for loop’, i.e. Line 13 of Listing 3.10, will generate error, if we execute the code using ‘gcc forloop.c -o out’, as this operation is only allowed with the C99 or C11 compiler. This code can be executed with command ‘gcc -std=c99 forLoop.c -o out’ or ‘g++ forloop.c -o out’.

Explanation Listing 3.10