Flow of control: if, if-else, if-else if-else statements

A programmer can decide which part of the code to execute based on some condition. The statements used for this are called conditional statements. These statements evaluate one or more conditions and make the decision whether to execute a block of code or not.

The `if` statement:

The `if` statement is the simplest form of decision-making. A general form of `if` statement is given by

```
 if (expr) {
// Block of code to be executed if expr is true
}
// Code continues here after the if block
```

It executes the block of code (enclosed by the curly braces `{}`) only if the specified `expr` evaluates to true (any non-zero value). If the `expr` is false (zero or null), the code block is skipped. *If the block contains only a single statement, the curly braces* `{}` *can be omitted, but it is best practice to include them for readability and to prevent errors.* Also, we can have `if` statement(s) in the block of code, i.e., nested `if` statements is allowed. Consider the program (given below) in which a real number is read. If it is nonnegative, its square root is calculated and printed. If it is negative, the program stops with an appropriate message.

```
#include<stdio.h>
#include<math.h>
int main()
{
double x,s;
printf("Enter a nonnegative numbers:");
scanf("%lf",&x);   // double x is read using %lf
if(x<0.0){         // start of if block
printf("Entered number must be nonnegative\n");
return 0;
}                 // end of if block
s=sqrt(x);
printf("Square root of %.2f is %.2f\n",x,s);
return 0;
}
```

Listing 16: C program "csqrt.c"

Compile with `$ gcc csqrt.c -lm ↩` and run the default executable with `$ ./a.out ↩`. The following shows the expected input and output of the program.

```
Enter a nonnegative numbers:2.5
Square root of 2.50 is 1.58
Enter a nonnegative numbers:-2.1
Entered number must be nonnegative
```

Note that we have two `return 0` expressions. The `return 0` expression at the top is reached when the expression `x<0` for the `if` statement is true. The execution of a C program stops whenever it executes a "`return 0;`" statement inside the main program.

The `if-else` statement:

The `if-else` statement is closely related to the `if` statement. It has the form

```
if (expr){
// code to be executed if the expr is true
}
else {
// code to be executed if the expr is false
}
```

It executes the block of code enclosed by the first curly braces `{}` only if the specified `expr` evaluates to true (any non-zero value). If the `expr` is false (zero or null), the code block enclosed by second curly braces `{}` is executed. Also, we can have `if` statement(s) or nested `if-else` statement(s) in the block of code. Consider the program (given below) in which two real numbers are read. Then it find their maximum and print it.

```
#include<stdio.h>
int main()
{
double x,y,max;
printf("Enter two real numbers:");
scanf("%lf%lf",&x,&y);
if(x>y){         // start of if-else block
max=x;
}
else {
max=y;
}                 // end of if-else block
printf("Maximum of %.2f and %.2f is %.2f\n",x,y,max);
return 0;
}
```

Listing 17: C program "cmax.c"

Compile with $ gcc cmax.c↩ and run the default executable with $ ./a.out ↩. The following shows the expected input and output of the program.

```
Enter two real numbers:2.5 7
Maximum of 2.50 and 7.00 is 7.00
```

Note that we can rewrite the program given in Listing 16 using `if-else` construct as follows:

```
#include<stdio.h>
#include<math.h>
int main()
{
double x,s;
printf("Enter a nonnegative numbers:");
scanf("%lf",&x);   // double x is read using %lf
if(x<0.0){         // start of if block
printf("Entered number must be nonnegative\n");
}
else {
s=sqrt(x);
printf("Square root of %.2f is %.2f\n",x,s);
}                // end of if-else block
return 0;
}
```

Listing 18: C program "csqrtm.c"

We again get the same output, but we use only one `return 0` expression.

The `if-else if-else` statement:

The general form of an `if-else if-if` ladder is shown below:

```
if (expr1) {
    // Code to be executed if expr1 is true
}
else if (expr2) {
    // Code to be executed if expr1 is false and expr2 is true
}
else if (expr3) {
    // Code to be executed if expr1 and expr2 are false and expr3 is true
```

```
}
    // .... Multiple else if blocks are possible
else {
    // Code to be executed if all exprs are false
}
```

Note that the `else if` and final `else` blocks are optional. The comments in the above explains the working of the `if-else if-else` ladder. Consider a program in which an arithmetic operator `op` and two integers `a, b` are read. The program then computes `a op b`, where `op` may be `+, -, *, /, %`. Finally it prints out the result.

```
#include<stdio.h>
int main()
{
char op;
int a,b,res;
printf("Enter arithmetic operator:");
scanf("%c",&op);
printf("Enter intgers a and b:");
scanf("%d%d",&a,&b);
if(op=='+')
   {
   res=a+b;
   }
else if(op=='-')
   {
   res=a-b;
   }
else if(op=='*')
   {
   res=a*b;
   }
else if(op=='/')
   {
       if(b==0)
       {
        printf("Divisor must be nonzero\n");
        return 0;
        }
   res=a/b;
```

```
   }
else if(op=='%')
   {
       if(b==0)
       {
       printf("b must be nonzero in a%%b\n");
       return 0;
       }
   res=a%b;
   }
else
   {
   printf("Unknown operator\n");
   return 0;
   }
printf("%d %c %d = %d\n",a,op,b,res);
return 0;
}
```

Listing 19: C program "carithm.c"

Compile with $ gcc carithm.c↩ and run the default executable with $ ./a.out ↩. The following shows the expected input and output of the program.

```
Enter arithmetic operator:+
Enter intgers a and b:4 5
4 + 5 = 9
Enter arithmetic operator:/
Enter intgers a and b:5 0
Divisor must be nonzero
Enter arithmetic operator:%
Enter intgers a and b:9 6
9 % 6 = 3
```

We have used nested `if` block in the above code. Note that the curly braces for a particular `if` block, `else if` block and `else` block are aligned along the same vertical column. This helps in visualizing the starting and ending for a particular block. Further, the operator can be written in any order. For example, we could have start the ladder with `%`, then `+`, then `/`, and so on. However, sometimes this kind of reordering is not possible. For example, consider the problem of determining division corresponding to marks obtained by a student. Marks greater than 66,

50 and 33 correspond to first, second and third division respectively. Any mark less than or equal to 33 represents fail. The following program implement this.

```
#include<stdio.h>
int main()
{
int marks;
printf("Enter marks:");
scanf("%d",&marks);
if(marks<0 || marks>100)
{
 printf("Invalid marks\n");
 return 0;
}

if(marks>66)
{
 printf("You have passed in the first division\n");
}
else if(marks>50)
{
  printf("You have passed in the second division\n");
}
else if(marks>33)
{
  printf("You have passed in the third division\n");
}
else
{
 printf("Sorry! You have failed\n");
}

return 0;
}
```

Listing 20: C program "mdiv.c"

Compile with `$ gcc mdiv.c↩` and run the default executable with `$ ./a.out ↩`. The following shows the expected input and output of the program.

```
Enter marks:105
```

```
Invalid marks
Enter marks:56
You have passed in the second division
Enter marks:30
Sorry! You have failed
```

Note that reordering of exprs in `if` and `else if` in the above code will lead to incorrect result. Finally, we write a program that accepts a year between 1000 and 2025 (both inclusive) and prints whether it is a leap year or not. We know that a leap year is divisible by 4 except for century years (years ending with 00). The century year is a leap year only if it is divisible by 400.

```
#include<stdio.h>
int main()
{
int year;
printf("Enter year between 1000 & 2025:");
scanf("%d",&year);
if(year<1000 || year>2025)
{
 printf("Invalid year\n");
 return 0;
}

if(((year%4==0) && (year%100!=0)) || (year%400==0))
{
 printf("Year %d is a leap year\n",year);
}
else
{
 printf("Year %d is not a leap year\n",year);
}

return 0;
}
```

Listing 21: C program "leap.c"

The following shows some expected input and output of the program:

```
Enter year between 2000 & 2025:900
```

```
Invalid year
Enter year between 2000 & 2025:1600
Year 1600 is a leap year
Enter year between 2000 & 2025:1700
Year 1700 is not a leap year
```

Note that the expression
`if(((year%4==0) && (year%100!=0)) || (year%400==0))`
in the code can be written as
`if(year%4==0 && year%100!=0 || year%400==0)` (*)
The reasons for this are the operator precedence and associativity. Since the equality operators have higher precedence than logical operators, the expression in (*) is equivalent to
`if((year%4==0) && (year%100!=0) || (year%400==0))`
Now, out of the logical operators, `&&` has higher priority than `||`. Hence, this is further equivalent to
`if(((year%4==0) && (year%100!=0)) || (year%400==0))`
which has been used in the code.

Return to Lecture Topics