International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 06 Issue: 06 | June 2019 www.irjet.net p-ISSN: 2395-0072

Switch Case Statements in C

Naren Khatwani

Student, Department of Computer Engineering, Vivekanand Education Society's Institute of Technology, Maharashtra, India

---

Abstract: *C programs are very versatile in nature. These programs handle the dynamic operations through a number of instructions and syntax. One such operation performed is the use of switch cases with appropriate break statements. However, it has been observed that the code repeats itself majority of times during the switch case statement execution which is a major drawback. This paper thus focusses on the mechanism to avoid repetition of code through the concept of fall through and expresses the methodology to achieve fall through using appropriate break statements and required syntax.*

1. INTRODUCTION

Languages used in any computer system has a set of instructions to execute. These languages though interpreted by the systems are in binary, assembly and higher level languages form the crux for any human being to operate the system. These high level languages have a group of instructions written to form the code that needs executions

It has been observed that though switch case operation is very versatile operation it has a number of drawbacks namely: repetition of code, increased execution time, reduced usability. Though it is widely accepted the drawbacks tend to decease the effectiveness of the language. One solution to resolve the conflicts created by switch case is using the mechanism of Fall through. Fall through in switch statement means jumping of statements during absence of break statement after each case.

1.1 SWITCH STATEMENT

The basic syntax of the switch case is as follows

switch (variable to be used) {

case name1 : statements;

break;

case name2 : statements;

break;

case name2 : statements;

break;

default : statements;

}

Where case indicates,

Break is used to come out of the switch case condition. It can be said that break is used after each switch case condition

Thus it can be said that switch case is used when there exist different conditions for one single expression. The syntax of switch statement above clearly shows the usage of break statement. We can see that break statement causes an immediate exit from the switch.

However as explained below fall through has the syntax

switch (variable to be used)

{

case name1 :

case name2 :

case name3 : statements;

break;

case name4 :

case name5 : statements;

break;

default : statements;

}

This indicates that fall through can be used when there exist some common conditions for a single expression while break statement is used only in the common statement condition for multiple switch cases and In fall through it can be clearly seen that a common statement can be written as a condition for multiple cases in order to provide reusability to the code.

Thus the difference in operations of both is as expressed in figure 1.Mode of Operation

A break is observed in figure 1 b at the end of all the statements. This break is necessary to be added after the last switch case because this will not cause an extra case to be executed by a mistake.

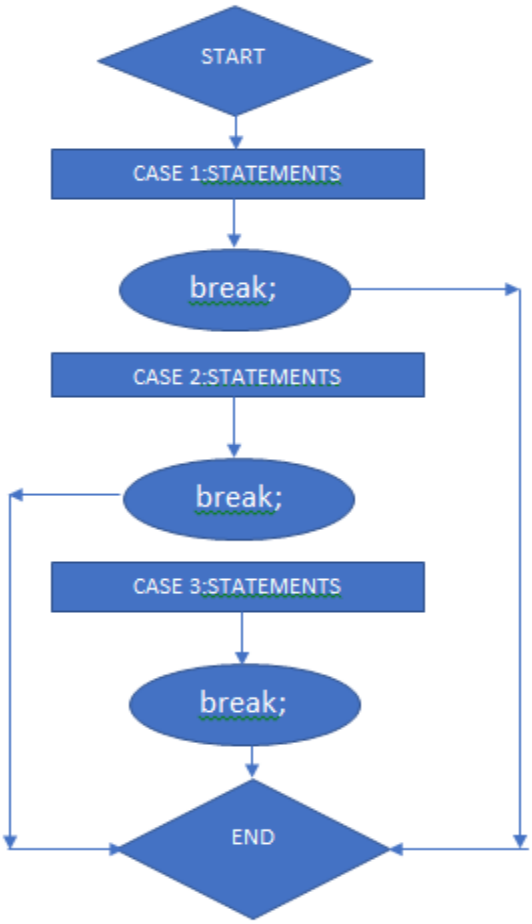

Figure 1(a):Switch Statement

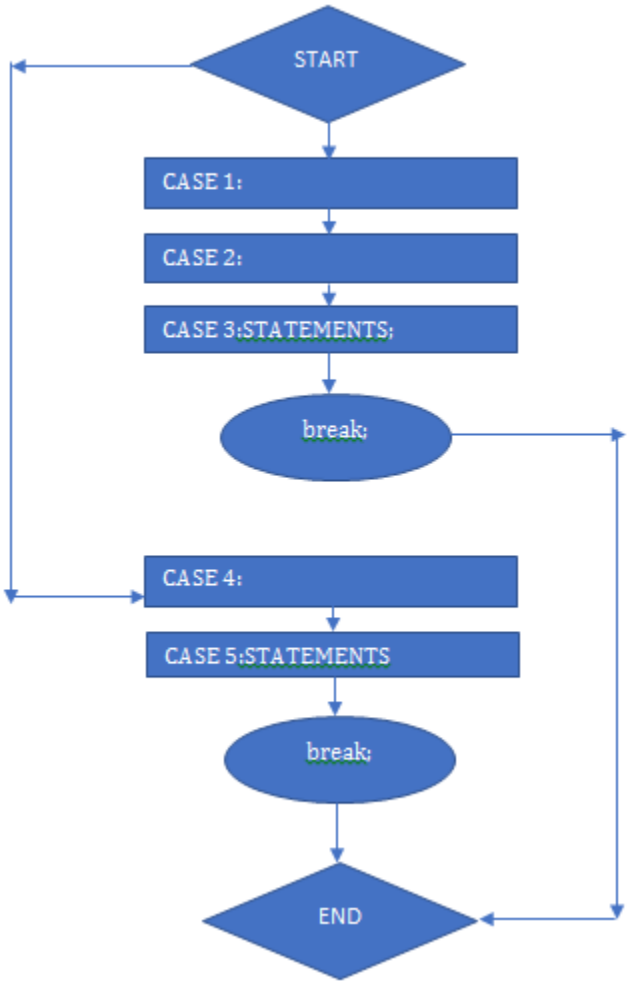

Figure 1(b):Switch Statement (using Fallthrough)

No of days in a month without fallthrough

```
#include<stdio.h>
int main()
 {
int n,t;
printf("ENTER THE NUMBER OF MONTH WHOSE NUMBER OF DAYS YOU WANT TO KNOW");
scanf("%d",&n);

switch(n)
{
case 1:printf("31 days");
break;
case 2:printf("ENTER YEAR");
scanf("%d",&t);

if(((t%4==0)||(t%400==0))&&(t%100!=0))
printf("29 DAYS");
else
printf("28 DAYS");
break;
```

```
case 3:printf("31 days");
break;
case 4:printf("30 days");
break;
case 5:printf("31 days");
break
case 6:printf("30 days");.....contd
```

NO OF DAYS IN A MONTH WITH FALL THROUGH

```
#include<stdio.h>
int main()
{
int n,t;
printf("ENTER THE NUMBER OF MONTH WHOSE NUMBER OF DAYS YOU WANT TO KNOW");
scanf("%d",&n);

switch(n)
{
case 1:
case 3:
case 5:
case 7:
case 8:
case 10:
case 12:printf("31 days");
break;
case 2:printf("ENTER YEAR");
scanf("%d",&t);
if(((t%4==0)||(t%400==0))&&(t%100!=0))
{
printf("29 DAYS");
}
```

FALL TRHOUGH OCCURS HERE WHEN THE USER ENTERS THE VALUE 1,3,5,7,8,10

```
else
{.....contd
```

1.2 Applications of Fallthrough

Duff's Device:

In the C programming language, Duff's device is a way of manually implementing loop unrolling by interleaving two syntactic constructs of C: the do-while loop and a switch statement.

Loop Unrolling:

Loop unrolling, also known as loop unwinding, is a loop transformation technique that attempts to optimize a program's execution speed at the expense of its binary size, which is an approach known as space–time tradeoff. The transformation can be undertaken manually by the programmer or by an optimizing compiler.

The following example explains the usage of Loop Unrolling:

Consider a normal for Loop that starts from the value j=5000 and decrements until the value of j is not equal to 0 and the statement inside the for loop adds a value s to the array element j with each iteration.

```
for(j=5000;j!=0;j--)
{
b[j]=b[j]+s;
}
```

Now in order for loop unrolling to occur we need to do some changes in the for loop

```
for(j=5000;j!=0;j=j-2)
{
b[j]=b[j]+s;
b[j-1]=b[j-1]+s;
}
```

Here we have added one more statement in the for loop which will do the same operation of adding a value 's' with each iteration to the (j)th element as well as for the (j-1)th element

Also we need to change the decrement condition to compensate the variable 's' we added for loop unrolling to occur.

As we can see above loop unrolling will cause the iterations to turn near to half the no of iterations that happened .

Working of Duffs Device:

First pass:

```
int count ;  // Set to 20

{

   int m = (count + 7) / 8;    // n is now 3.  (The "while" is going to be run three times.)

   switch (count % 8) {          // The remainder is 4 (20 modulo 8) so jump to the case 4

case 0:             // [skipped]

do {                // [skipped]

*to = *from++;    // [skipped]

case 7:*to = *from++;   // [skipped]

case 6:*to = *from++;   // [skipped]

case 5:*to = *from++;   // [skipped]

case 4:*to = *from++;    // Start here.  Copy 1          byte (total 1)

case 3:*to = *from++;   // Copy 1 byte (total 2)

case 2:*to = *from++;   // Copy 1 byte (total 3)

case 1:*to = *from++;   // Copy 1 byte (total 4)

} while (--m > 0);     // M = 3 Reduce M by 1, then jump up

//to the "do" if it's still

}                    // greater than 0 (and it is)

}
```

Second Pass:

```
int count;

{

   int m = (count + 7) / 8;

   switch (count % 8) {

   case 0:

   do {              // The while jumps to here.

   *to = *from++;   // Copy 1 byte (total 5)

  case 7:*to = *from++;   // Copy 1 byte (total 6)

  case 6: *to = *from++;   // Copy 1 byte (total 7)

  case 5:*to = *from++;   // Copy 1 byte (total 8)

  case 4:*to = *from++;   // Copy 1 byte (total 9)

  case 3:*to = *from++;   // Copy 1 byte (total 10)

  case 2:*to = *from++;   // Copy 1 byte (total 11)

  case 1:*to = *from++;   // Copy 1 byte (total 12)

   } while (--m > 0);      // M = 2 Reduce M by 1, then jump up to the "do" if it's still

   }   // greater than 0 (and it is)

}
```

Third Pass:

```
int count;

{

  int m = (count + 7) / 8;

  switch (count % 8) {

  case 0:

  do {              // The while jumps to here.

 *to = *from++;   // Copy 1 byte (total 13)

  case 7:*to = *from++;   // Copy 1 byte (total 14)

  case 6:*to = *from++;   // Copy 1 byte (total 15)

  case 5:*to = *from++;   // Copy 1 byte (total 16)
```

```
  case 4:*to = *from++;   // Copy 1 byte (total 17)

  case 3:*to = *from++;   // Copy 1 byte (total 18)

  case 2:*to = *from++;   // Copy 1 byte (total 19)

  case 1:*to = *from++;   // Copy 1 byte (total 20)

  } while (--m > 0);     // M = 1  Reduce M by 1, then jump up to the "do" if it's still

 }  // greater than 0 (and it's not, so bail)

} // continue here...
```

Conclusions:

C programs are considered very versatile. However in order to avoid repetition of code two mechanisms namely switch statement and break can be employed. Fallthrough as a concept can enable repetition of code through expressions.

References:

[1]. Holly, Ralf. "A reusable Duff device." *Dr. Dobb's Journal* 30.8 (2005): 73-74.

[2]. The C Programming Language-Book by Brian Kernighan and Dennis Ritchie

[3]. The C programming Language By Brian W. Kernighan and Dennis M. Ritchie. Published by Prentice-Hall in 1988 ISBN 0-13-110362-8 (paperback) ISBN 0-13-110370-9

[4]. Expert C Programming: Deep C Secrets Book by Peter van der Linden

[5].https://en.wikipedia.org/wiki/Duff%27s_device

[6]. https://en.wikipedia.org/wiki/Loop_unrolling

[7]. LET US C SOLUTIONS -15TH EDITION Book by Yashavant Kanetkar

[8].21st Century C: C Tips from the New School Book by Ben Klemens

[9]. Head First C-Book by David Griffiths and Dawn Griffiths