Expression & Statement (Command)

- A pure expression gives a value e.g. 2, -2, $-a$, $-2 * a + b$.
- A command or a statement changes the content of a location, but does not have a value.
- In C language many expressions are impure and cause side effects by changing value of locations e.g. `++count`, n = 2*m + 4.

- Any expression in C (with or without any side effect) can be converted to a statement by putting a semicolon at the end[a]. These are called expression statements.
- This blurs the distinction between an expression and a command in this language.

[a] There is no value of a statement.

Null Statement

- A semicolon in C language is used as a statement terminator.
- The semicolon ‘;’ itself is viewed as null statement (no effect).

Compound Statement

- A sequence of statements within a pair of curly braces { } forms a single compound statement or block.
- Variables can be declared within a block, and they are local to the block (visible only within the block).
- A name clash is resolved in favor of the local names.

```
#include <stdio.h>
int main() // temp17.c
{
  int a = 10, b = 20, c = 30;
  {
    int b = 200, c = 300 ;
    {
      int c = 3000 ;
      printf("L3 - a: %d, b: %d, c: %d\n",
               a, b, c) ;
    }
  printf("L2 - a: %d, b: %d, c: %d\n",
       a, b, c) ;
 }
 printf("L1 - a: %d, b: %d, c: %d\n",
        a, b, c) ;
 return 0 ;
}
```

```
$ cc -Wall temp17.c
$ ./a.out
L3 - a:  10, b:  200, c:  3000
L2 - a:  10, b:  200, c:  300
L1 - a:  10, b:  20, c:  30
```

Change in Control Flow

- It may be necessary to perform different sets of operations in a program, depending on data.
- This calls for data dependent choice of the execution sequence of statements - control-flow.

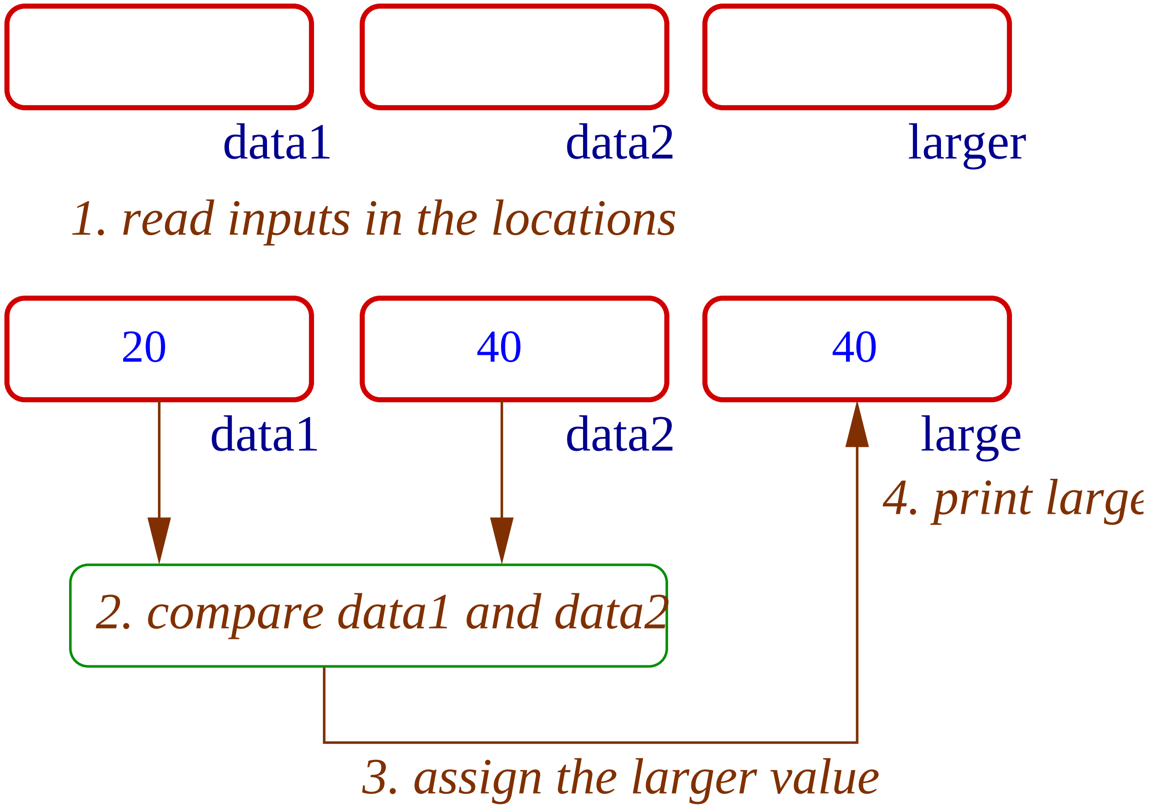
Example I

Write a C Program that reads two `int` data from the keyboard, finds the larger among them, and prints it on the VDU.

Sequence of Operations

1. Read the two input data in two variables (locations) of type `int`.
2. Compare the variables (r-values) and put the larger value in a third variable (location).
3. Print the content of the third variable.

int data1, data2, larger ;



data1 data2 larger

1. read inputs in the locations

20 40 40

data1 data2 large

4. print large

2. compare data1 and data2

3. assign the larger value

```
#include <stdio.h>
int main() // temp18.c
{
    int data1, data2, larger;
    printf("Enter two integer data: ");
    scanf("%d%d", &data1, &data2);
    if(data1 > data2) larger = data1 ;
    else larger = data2;
    printf("\n%d is the larger among %d & %d\n",
                larger, data1, data2);
    return 0 ;
}
```

if Statement

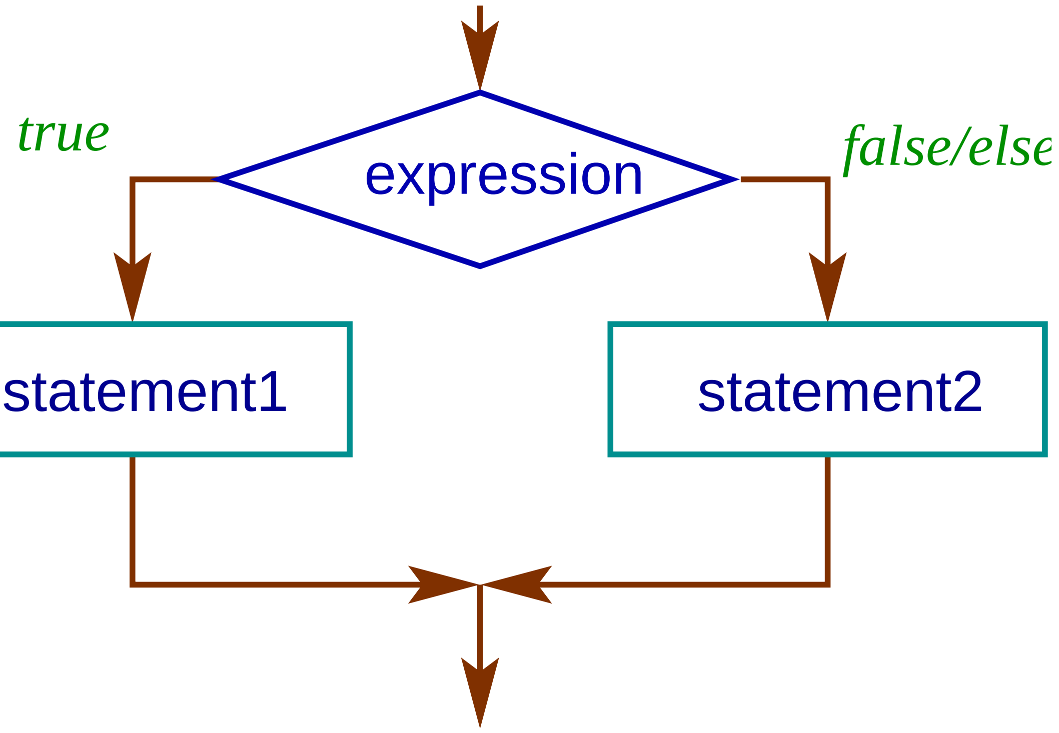
A command called if-statement is used for controlling the execution sequence in this program. The structure or syntax of if-statements are as follows.

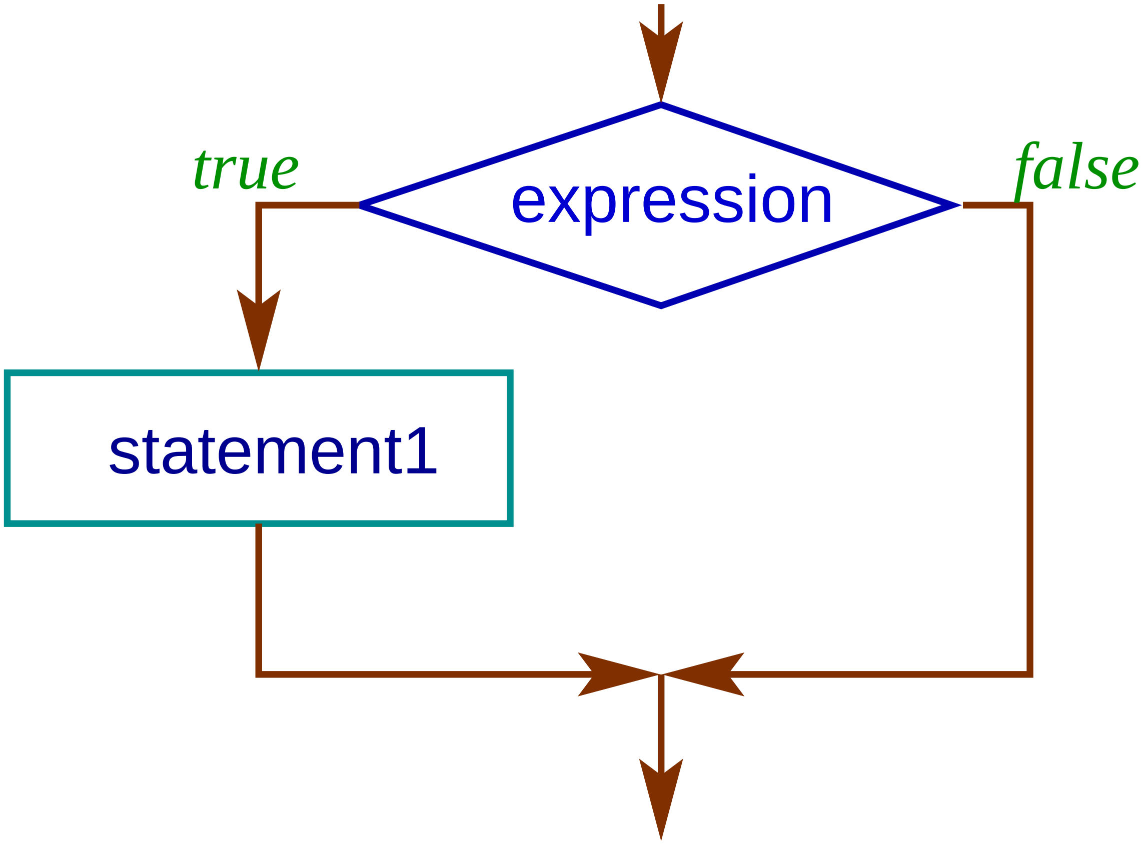
if (expression) statement_1 else statement_2

if (expression) statement_1

In this example we use the first type and we have

- expression: `data1` $>$ `data2`
- statement$_1$: `larger = data1;`
- statement$_2$: `larger = data2;`


true
expression
false/else
statement1
statement2


true
expression
false
statement1

Relational and Boolean Expressions

Two new types of expressions are used in `if`-statement and other control-flow constructs of the language. They are called relational and boolean expressions.

Relational and Boolean Expressions

C language does not have distinct truth values (true and false). The value zero (0) is treated as false and any non-zero value is treated as true.

```
#include <stdio.h>
int main() // temp19.c
{
    int a;

    scanf("%d", &a) ;
    if(a) printf("non-zero\n") ;
    else printf("zero\n") ;
    return 0 ;
}
```

```
$ cc -Wall temp19.c
$ ./a.out
0
zero
$ ./a.out
-1
non-zero
$ ./a.out
1
non-zero
```

Relational Operators

Following are the relational operators with their usual meaning.

$==$ (equal to), $! =$ (not-equal to), $<$ (less than) $>$ (greater than), $<=$ (less than or equal to), $>=$ (greater than or equal to).

The usual operands of relational operators are `int`, `float`, `char` etc. Their values are boolean.

Boolean Operators

Following are the boolean operators with their usual meaning.

&& (logical and), ∼ (logical not), || (logical or).

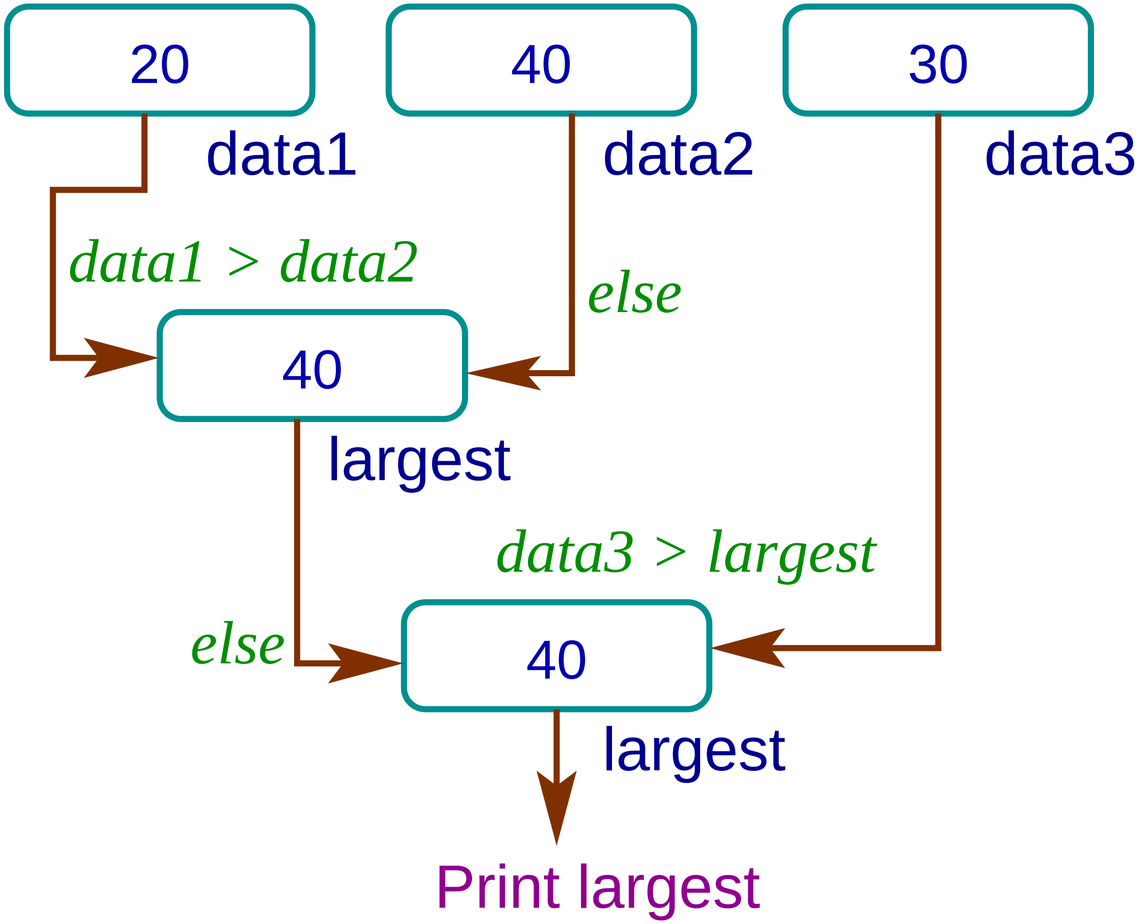
The operands and values of boolean operators are boolean values. Find out the precedence and associativity of these operators from the book.

Example II

Write a C program to find the largest among three `int` data.

Sequence of Operations

1. Read three input integers in three variables `data1`, `data2` and `data3` of type `int`.
2. Compare `data1` and `data2`, put the larger value in a fourth variable, `largest`.
3. Compare `data3` and `largest`. If `data3` is larger, copy it in `largest`.
4. Print the content of `largest`.


20
40
30
data1
data2
data3
data1 > data2
else
40
largest
data3 > largest
else
40
largest
Print largest

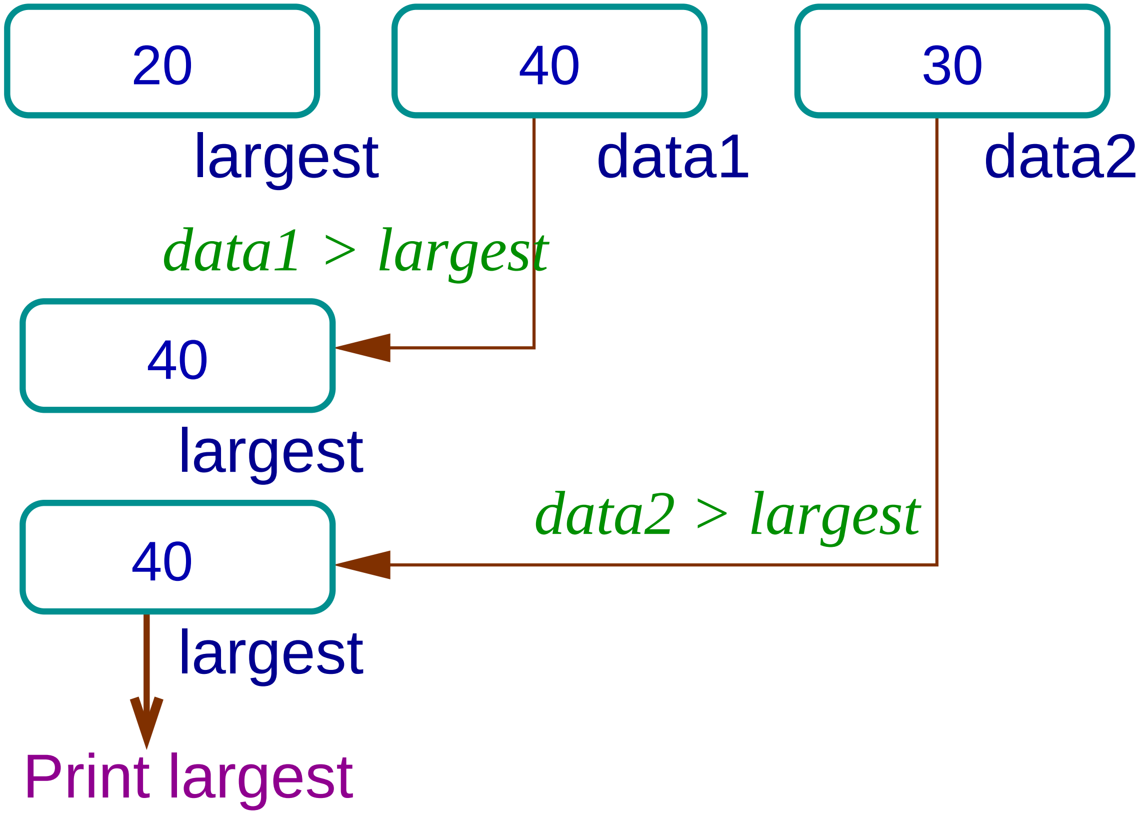
```
#include <stdio.h>
int main() // temp20.c
{
  int data1, data2, data3, largest ;
  printf("Enter three integer data: ") ;
  scanf("%d%d%d", &data1, &data2, &data3) ;
  if(data1 > data2) largest = data1 ;
  else largest = data2 ;
  if(data3 > largest) largest = data3 ;
  printf("\n%d is the largest among %d, %d & %d\n",
            largest, data1, data2, data3);
  return 0;
}
```

An Alternate Sequence

1. Read the first input in the variable `largest`.
2. Read the second and third data in `data1` and `data2`.
3. If `data1` is greater than `largest`, copy `data1` to `largest`.
4. If `data2` is greater than `largest`, copy `data2` to `largest`.
5. Print the content of `largest`.

Note

In this method we use three variables but one input data may be lost at the end.


20
largest
40
data1
30
data2
data1 > largest
40
largest
data2 > largest
40
largest
Print largest

```
#include <stdio.h>
int main() // temp21.c
{
  int data1, data2, largest ;
  printf("Enter three integer data: ") ;
  scanf("%d%d%d", &largest, &data1, &data2) ;
  if(data1 > largest) largest = data1 ;
  if(data2 > largest) largest = data2 ;
  printf("\n%d is the largest data\n",
            largest);
  return 0;
}
```

Alternate Sequence

1. Read the first input in the variable `largest`.

2. Read the second data in `data`.

3. If `data` $>$ `largest`, copy `data` to `largest`.

4. Read the third data in `data`.

5. If `data` $>$ `largest`, copy `data` to `largest`.

6. Print the content of `largest`.

```
#include <stdio.h>
int main() // temp21a.c
{
    int data, largest ;
    printf("Enter 3 integers : ") ;
    scanf("%d%d", &largest, &data) ;
    if(data > largest) largest = data ;
    scanf("%d", &data) ;
    if(data > largest) largest = data ;
    printf("\n%d is the largest data\n",
            largest);
    return 0;
}
```

Note

We use two variables, so two input data may be lost at the end.

Alternate Sequence

1. Read three input in `data1`, `data2` and `data3`.
2. If `data1` $>$ `data2`, if `data1` $>$ `data3`, then `data1` contains the largest value.
3. Similarly consider the other cases.

```
#include <stdio.h>
int main() // temp21b.c
{
    int data1, data2, data3, largest ;
    printf("Enter three integer data: ") ;
    scanf("%d%d%d", &data1, &data2, &data3) ;
    if(data1 > data2)
       if(data1 > data3) largest = data1 ;
       else largest = data3 ;
    else if(data2 > data3) largest =data2 ;
         else largest = data3 ;
    printf("\n%d is the largest data\n", largest);
    return 0;
}
```

Note

This is an example of nested if-statement. No input data is lost in this case.

Note

Statements within the if par and the else part of an if-statement, may be compound statements.

```
if (expression) {
                    $\text{statement}_1$
                    ...
                    $\text{statement}_k$
                }
else            {
                    $\text{statement}_1$
                    ...
                    $\text{statement}_m$
                }
```

```
if (expression) {
                statement$_1$
                . . .
                statement$_k$
            }
```

```
#include <stdio.h>
int main() // temp22.c
{
    int data;
    printf("Enter an integer: ") ;
    scanf("%d", &data) ;
    if (data<0) printf("-ve\n");
    else if (data == 0) printf("zero\n");
         else printf("+ve\n") ;
    return 0;
}
```

Associating `else`

We know that `if` statements can be nested. The `else` part will be associated to the nearest `if` by default. It is better to use curly braces {} to disambiguate the association.

```
#include <stdio.h>
int main() // temp23.c
{
    int data;
    printf("Enter an integer: ") ;
    scanf("%d", &data) ;
    if (data>0)
        if (data%5) printf("not-divisible\n");
    else printf("-ve data\n"); // incorrect assoc
    return 0;
}
```

```
$ cc -Wall temp23.c
temp23.c:  In function ‘main’:
temp23.c:7:  warning:  suggest explicit braces
to avoid ambiguous ‘else’
$ ./a.out
Enter an integer:  -3
$ ./a.out
Enter an integer:  3
not-divisible
$ ./a.out
Enter an integer:  10
-ve data
```

```
#include <stdio.h>
int main() // temp23a.c
{
    int data;
    printf("Enter an integer: ") ;
    scanf("%d", &data) ;
    if (data>0) {
        if (data%5) printf("not-divisible\n");
    }
    else printf("-ve data\n");
    return 0;
}
```

```
$ cc -Wall temp23a.c
$ ./a.out
Enter an integer:  -3
-ve data
$ ./a.out
Enter an integer:  3
not-divisible
$ ./a.out
Enter an integer:  10
$
```

switch Statement

It may be necessary to take multi-way decision and control of execution in a program.

C language uses `switch` statement where the control is transfered by matching the value of an expression to a value from a finite set of constants.

switch Statement

switch (expression) {

case const-exp$_1$: statement$_1$

case const-exp$_2$: statement$_2$

. . .

case const-exp$_k$: statement$_k$

default: statement$_{k+1}$

}

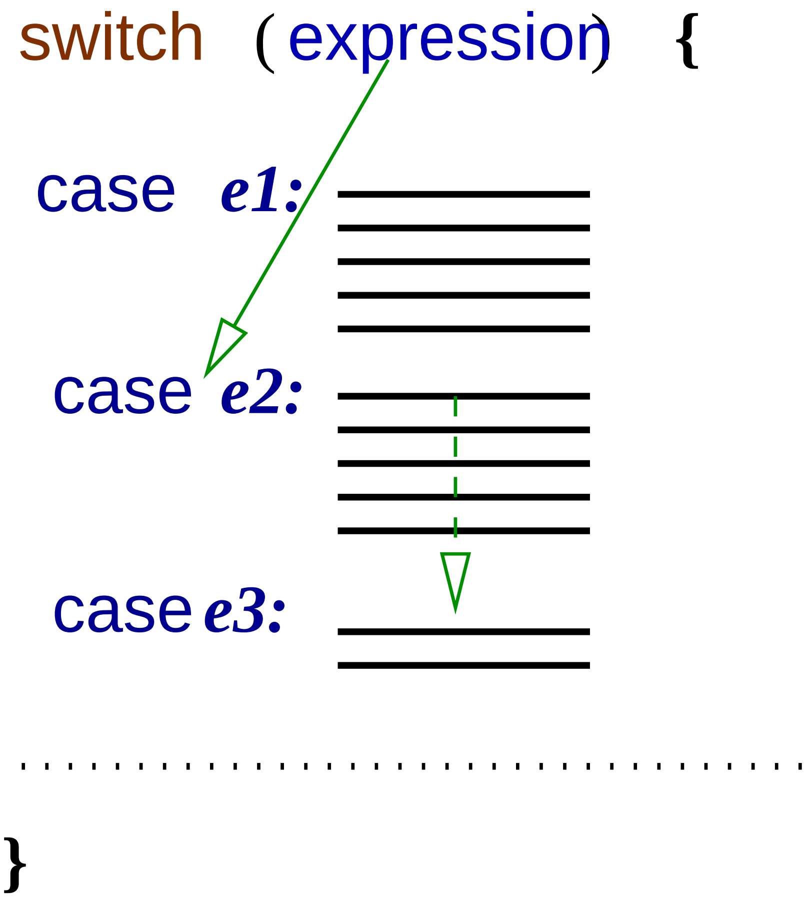
Example III

Read a non-negative integer and take different actions depending on the remainders obtained dividing the data by 5.

```
#include <stdio.h>
int main() { // Incorrect, temp24.c
    int data;
    printf("Enter a +ve integer: ") ;
    scanf("%d", &data) ;
    switch(data%5){
           case 0: printf("remainder is 0\n") ;
           case 1: printf("remainder is 1\n") ;
           case 2: printf("remainder is 2\n") ;
           case 3: printf("remainder is 3\n") ;
           default: printf("remainder is 4\n") ;
    }
    return 0;
}
```

```
$ cc -Wall temp24.c
$ ./a.out
Enter a +ve integer:  27
remainder 2
remainder 3
remainder 4
```

The control is falling through. It is to be transfered out of the `switch` statement.



switch (expression) {

case *e1:*

case *e2:*

case *e3:*

. . . .

}

break Statement

A break statement forces the control out of the switch statement.

```
#include <stdio.h>
int main() { // temp25.c
    int data;
    printf("Enter a +ve integer: ") ;
    scanf("%d", &data) ;
    switch(data%5){
       case 0: printf("remainder 0\n"); break;
       case 1: printf("remainder 1\n"); break;
       case 2: printf("remainder 2\n"); break;
       case 3: printf("remainder 3\n"); break;
       default: printf("remainder 4\n");
    }
    return 0;
}
```

```
$ cc -Wall temp25.c
$ ./a.out
Enter a +ve integer:  27
remainder 2
```