Performing ``Non-Local'' Goto

In many situations, you may want to jump to a location that is very far away from the current function. Sometimes, the destination is even not in the same function. For example, when a signal occurs, the program logic may require the control flow be sent to a very special function after the signal is handled. In this case, some addition mechanism is required to perform this type of non-local gotos. This is the job of header file **setjmp.h** and functions **setjmp()** and **longjmp()**.

To perform non-local gotos, you need to do the following:

- Include the header file **setjmp.h**
- You have to use two functions: **setjmp()** for setting a return point and **longjmp()** to perform a goto.
- Declare a variable of type **jmp_buf**. This is called a **jump buffer**.
- Here is how to do the job:
  - Declare a variable of type **jmp_buf**

    ```
    jmp_buf JumpBuffer;
    ```

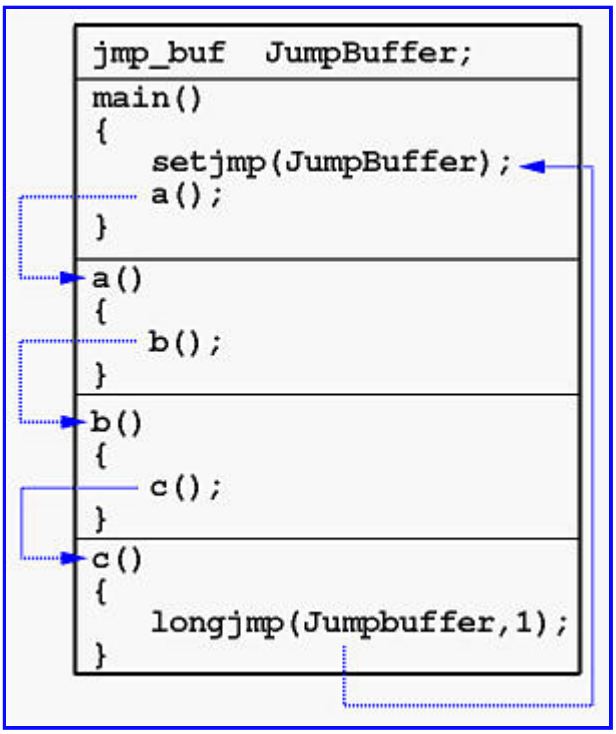
    This jump buffer is used to store the ``environment'' of a return point in your program.
  - Mark a ``return'' point using function **setjmp()**. This function has only one argument of type **jmp_buf**. Then, the current state of program execution at that point is saved.

    ```
    jmp_buf JumpBuffer;

    setjmp(JumpBuffer);
    ```

    The above saves the current state of program execution at that point to the jump buffer **JumpBuffer**.
  - Later on, in some other functions (or even in the same function), you can use **longjmp()** to jump back to the marked return point. **longjmp()** takes two arguments. The first is a jump buffer and the second is a non-zero integer.

    ```
    jmp_buf JumpBuffer;

    longjmp(JumpBuffer, 1);
    ```

    In this case, **longjmp()** sends the control back to the place where the jump buffer **JumpBuffer** receives its most recent information.
  - This activity can be illustrated as follows:



The main program sets a jump buffer with **setjmp()** and then calls **a()**. In **a()**, it calls **b()** which, in turn, calls **c()**. In function **c()**, since it has a **longjmp()** call with jump buffer **JumpBuffer**, the control will be sent back to the place where **JumpBuffer** receives its value. As a result, all function returns are by–passed.

Now you see the meaning of ***non–local***. The **longjmp()**executes a goto across several functions!

In the figure, the blue dot arrows indicate program execution flow.

But, You Need to Know More .....

You need to know more before start writing programs with non–local gotos:

- When **setjmp()** is called, it saves the current state of execution and **returns a zero**.
- When **longjmp()** is called, it sends the control back to a marked return point (a **setjmp()** call) and **lets setjmp() return the second argument of longjmp()**. Confused? Here is an example:

```
#include  <stdio.h>
#include  <setjmp.h>

jmp_buf   Jump;

void  A(...)
{
     if (setjmp(Jump) == 0) {
          printf("A return point marked\n");
          .....
     }
     else {
          printf("Just returned from a long journey\n");
          .....
     }
```

```
}

void  B(....)
{
     .....
     longjmp(Jump, 1);
     /* cannot reach here */
}
```

In function **A()**, when **setjmp(Jump)** is executed the first time, it returns 0 and marks a return point. The **if** statement tests the return value (in this case, 0) and prints a message. Sometime later, function **B()** is called and executes **longjmp(Jump,1)** with the second argument being 1. Then, the control is sent back to the place where the content of **Jump** is saved. This is, of course, the **setjmp(Jump)** call in function **A()**. Since this is a return from a **longjmp()** call, the value of the second argument of **longjmp()** is considered to be the return value of **setjmp()**. In this case, when the second time the control returns to **setjmp()**, its return value is 1 which forces the execution enters the **else**part.

The following figure illustrates this flow of control. The first time the control flow passes through **setjmp()**, jump buffer **Jump** is set and **setjmp()** returns 0. Therefore, the control flow follows the red arrow. The second time the control flow passes through **setjmp()**, which is due to the execution of the **longjmp(Jump,1)**in function **B()**, the return value of **setjmp()** is 1, the second argument of the call to **longjmp(Jump,1)**. This brings the control flow to the **else** part. In the figure, this is represented with blue dot arrow.