# C++ SIGNAL HANDLING

Signals are the interrupts delivered to a process by the operating system which can terminate a program prematurely. You can generate interrupts by pressing Ctrl+C on a UNIX, LINUX, Mac OS X or Windows system.

There are signals which can not be caught by the program but there is a following list of signals which you can catch in your program and can take appropriate actions based on the signal. These signals are defined in C++ header file <csignal>.

| Signal | Description |
|---|---|
| SIGABRT | Abnormal termination of the program, such as a call to **abort** |
| SIGFPE | An erroneous arithmetic operation, such as a divide by zero or an operation resulting in overflow. |
| SIGILL | Detection of an illegal instruction |
| SIGINT | Receipt of an interactive attention signal. |
| SIGSEGV | An invalid access to storage. |
| SIGTERM | A termination request sent to the program. |

## The signal function:

C++ signal-handling library provides function **signal** to trap unexpected events. Following is the syntax of the signal function:

```
void (*signal (int sig, void (*func)(int)))(int);
```

Keeping it simple, this function receives two arguments: first argument as an integer which represents signal number and second argument as a pointer to the signal-handling function.

Let us write a simple C++ program where we will catch SIGINT signal using signal function. Whatever signal you want to catch in your program, you must register that signal using **signal** function and associate it with a signal handler. Examine the following example:

```
#include <iostream>
#include <csignal>

using namespace std;

void signalHandler( int signum )
{
    cout << "Interrupt signal (" << signum << ") received.\n";

    // cleanup and close up stuff here
    // terminate program

   exit(signum);

}

int main ()
{
    // register signal SIGINT and signal handler
    signal(SIGINT, signalHandler);

    while(1){
```

```
        cout << "Going to sleep...." << endl;
        sleep(1);
    }

    return 0;
}
```

When the above code is compiled and executed, it produces the following result:

```
Going to sleep....
Going to sleep....
Going to sleep....
```

Now, press Ctrl+c to interrupt the program and you will see that your program will catch the signal and would come out by printing something as follows:

```
Going to sleep....
Going to sleep....
Going to sleep....
Interrupt signal (2) received.
```

## The raise function:

You can generate signals by function **raise**, which takes an integer signal number as an argument and has the following syntax.

```
int raise (signal sig);
```

Here, **sig** is the signal number to send any of the signals: SIGINT, SIGABRT, SIGFPE, SIGILL, SIGSEGV, SIGTERM, SIGHUP. Following is the example where we raise a signal internally using raise function as follows:

```
#include <iostream>
#include <csignal>

using namespace std;

void signalHandler( int signum )
{
    cout << "Interrupt signal (" << signum << ") received.\n";

    // cleanup and close up stuff here
    // terminate program

  exit(signum);

}

int main ()
{
    int i = 0;
    // register signal SIGINT and signal handler
    signal(SIGINT, signalHandler);

    while(++i){
        cout << "Going to sleep...." << endl;
        if( i == 3 ){
            raise( SIGINT);
        }
        sleep(1);
    }

    return 0;
}
```

When the above code is compiled and executed, it produces the following result and would come

out automatically:

```
Going to sleep....
Going to sleep....
Going to sleep....
Interrupt signal (2) received.
```