Exception Semantics

- What are exceptions?

- Why exception handling is important?
Exception Semantics

An exception is an error condition occurring from an operation that cannot be resolved by the operation itself.

🔹 What are exceptions?
  - Divide by zero, open a non-exist file, index out of range, …

🔹 Why exception handling is important?
  - Robustness
Ways to Handle Exceptions

✧ Have each function/method **returns a specific value**

✧ Have each function/method **set the value of a specific variable**

✧ **Try/Catch** approach: encapsulates code that could cause an exception in a try block. If an exception occurs, execution in the try block halts and restarts with the first statement in the catch block

• **Throw**: permits the programmer to throw an exception deliberately
#include <cstdio>

int main (int argc, char *argv[]) {
    for (;;) {
        int q;

        try {
            printf("Enter a number (0..9): ");
            int k = scanf("%d", &q);

            if (k == 0) {
                scanf("%*s"); //read the value but ignore
                throw "value is not a number";
            }

            if (q < 0 || q > 9) {
                throw q;
            }
        }
        catch (const char *s) {
            printf("Error: %s\n", s);
            continue;
        }
        catch (int v) {
            printf("Error: number %d is out of range (0..9)\n", v);
            continue;
        }
        if (q == 0)
            break;
    }
    return 0;
}
#include <cstdio>

int childFunc () {
    int q;
    try {
        printf("Enter a number (0..9): ");
        int k = scanf("%d", &q);
        if (k == 0) {
            scanf("%*s");
            throw "value is not a number";
        }
        if (q < 0 || q > 9) {
            throw q;
        }
    } catch (int v) {
        printf("Error: number %d is out of range (0..9)\n", v);
        return v;
    }
    return q;
}

int main () {
    int val;
    for (;;) {
        try {
            val = childFunc();
        } catch (const char *s) {
            printf("Error: %s\n", s);
            continue;
        }
        if (val == 0)
            break;
    }
    return 0;
}
import java.io.IOException;
import java.io.BufferedReader;
import java.io.InputStreamReader;

public class trycatch {

    public static void main (String[] args) throws IOException {
        int number;

        while (true) {
            try {
                BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
                System.out.print("Enter number: ");
                number = Integer.parseInt(in.readLine());
                break;
            }
            catch (NumberFormatException e) {
                System.out.println("Illegal number");
            }
            finally {
                System.out.println("In the finally block");
            }
        }
        System.out.printf("number is %d\n", number);
    }
}
```c
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>

int quit = 0;

void handler (int signal) {
    printf("Caught signal %d\n", signal);
    quit = 1;

    return;
    //exit(-1); //to terminate
}

int main () {
    signal(SIGINT, handler);

    while (!quit) {
        printf("blah\n");
    }

    printf("Cleaning up\n");

    return 0;
}
```
Discussions

- In C, exceptions can be handled by raising signals.
  - Signals are events raised by OS or programs to indicate an error has occurred. They will interfere with the normal program flow. But they are not for error catching. The program will attempt to return to execute from where it left off.

- In C++ and Java, exceptions can be handled by try/catch.
  - The program does not try to return to pick up from where it left off. Instead, execution moves to the catch block.