Semantics (XI)

Exception Semantics

- Error Handling
  - Function **return value**
    - **C** has the **convention** that a function that executes properly should return a 0 if it is not using the function’s return value for something else.
  - **Assertions.** Check the correctness of assumptions during run-time
  - **Signals.** A software interrupt delivered to a process. OS uses it to report exceptional situations to an executing program (e.g., references to invalid memory addresses).
  - **Exceptions are thrown/caught.** This disrupts the normal flow of execution.

- Assertions
  - Statements used to **check the correctness of assumptions** made in a program during runtime.
  - Debug tool

- C example:

```c
void assert (int expression);
```

- If the expression meaning is 0 (false), the expression, source code filename, and line number are sent to standard error, and then the abort function is invoked.

```c
// GPACalc.c
#include <stdio.h>
#include <assert.h>

float GPACalc (int size, float *ary) {
    float sum = 0;
    for (int i = 0; i < size; i++) {
        // check invalid gpa
        assert (ary[i] >= 0.0 && ary[i] <= 4.3);
        sum += ary[i];
    }
    return sum/size;
}

int main () {
    float a[] = {3.5, 3.8, 4.0, 4.1, 4.5};
    printf("%f\n", GPACalc(5, a));
    return 0;
}
```

```bash
mbsp-190250:assert yingli$ gcc GPACalc.c
mbsp-190250:assert yingli$ ./a.out
Assertion failed: (ary[i] >= 0.0 && ary[i] <= 4.3), function GPACalc, file GPACalc.c, line 9.
Abort trap: 6
mbsp-190250:assert yingli$ 
```
• Java example:

```java
assert expression;
assert expression1 : expression2;
```
- expression1: boolean expression; If false, JVM throws AssertionError.
- expression2: an expression that has a value.
- `java -ea`: enable assertions

```java
public class GPACalc {
    public static double calc (double ary[]) {
        double sum = 0;
        for (double f : ary) {
            assert f >= 0.0 && f <= 4.3 : "Invalid value " + f;
            sum += f;
        }
        return sum / ary.length;
    }

    public static void main (String args[]) {
        double a[] = {3.5, 3.8, 4.0, 4.1, 4.5};
        System.out.println(calc(a));
    }
}
```

• Python example

```python
assert condition, error_message(optional)
```
- condition: boolean expression
- error_message: printed on terminal if AssertionError

```python
# GPACalc.py
def GPACalc (ary):
    sum = 0.0
    for f in ary:
        assert f >= 0.0 and f <= 4.3, "Invalid value " + str(f)
        sum += f;
    return sum / len(ary)

def main ():
    a = [3.5, 3.8, 4.0, 4.1, 4.5]
    print(GPACalc(a))

if __name__ == "__main__":
    main()
```
Signals

- In C, signals can be used to handle exceptions, and it has a library to handle signals. These are not the same as exceptions (though they could be caused by similar errors, such as dereferencing a null pointer).
- An example of a signal that is not an exception is a key-board interrupt. This is an external signal and it needs to be handled.

- Show signal-int.c, run code and press ctl+c

```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 () {
    // SIGINT "program interrupt" signal, the signal is sent when the user type
    // INTR character, usually Ctrl+c
    signal(SIGINT, *handler);
    while (!quit) {
        printf("blah\n");
    }
    printf("Cleaning up\n");
    return 0;
}
```

- If the signal handler simply returns instead of exiting the program, the main program will continue where it left off. But it's not always possible. [comment out the exit(-1) in the above and remain return in the handler function, and run the code]

```c
Caught signal 2
Cleaning up
```
• Show signal-bus.c [if use return in the handler, it will generate infinite loop]

```c
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <string.h>

void bushandler( int signal ) {
    printf("Caught bus error %d\n", signal);
    // return; // infinite loop
    exit(-1); // gracefully leave program
}

int main(int argc, char *argv[]) {
    signal( SIGBUS, bushandler );
    // on Mac OS, the literal strings are stored in constant memory, so
    // we can't write to them.
    strcpy("hello", "goodbye");
    return 0;
}
```

• Show signal-segv.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>

float *x = NULL;
// assume global variable x has been allocated. print its value.
void do_something() {
    printf("in do_something\n");
    // the error happens inside printf
    printf("%.2f\n", *x);
}

void seghandler(int signal) {
    printf("Caught seg fault %d\n", signal);
    exit(-1);
}

void seghandler2(int signal) {
    printf("Caught seg fault %d\n", signal);
    x = &a;
    do_something();
    exit(0); // fixed the problem, restarted the
    // program, and leave with a success flag.
}

int main(int argc, char *argv[]) {
    signal(SIGSEGV, seghandler);
    do_something();
    return 0;
}
```
Exceptions
- The formal semantics of exceptions is out of the scope of this course.
- We will describe the flow of execution when an exception is encountered.

- Java example: **A function can throw an exception it does not catch.** To do this, it must declare the property of the exception to calling functions by including a throw clause in its preamble.
  - Like C++, **Java permits multiple catch blocks differentiated by their argument.**
  - The try/catch structure can also include a **block labeled with finally**, which has no arguments. **Code in the finally section is always executed** whether or not the code throws an exception. It is executed even if one of the try/catch blocks calls break or return. [Show trycatch.java, ask the outputs]

```java
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());
                if (number == 0) 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);
    }
}
```

$ javac trycatch.java
$ java trycatch
Enter number: 1
In the finally block
Enter number: 2
In the finally block
Enter number: 3
In the finally block
Enter number: 4
In the finally block
Enter number: 0
In the finally block
number is 0