# Introduction to Assertions in Programming

CS 152 Lecture

R. K. Joshi
IIT Bombay

# How do we know our program is correct?

We prove it on paper

What about the implementation?
can it carry part of the proof?

To work as Defense against errors

or to aid the development of the program?

# A Simple Idea

Use Assertions

A condition which should hold true where it is placed

**assert (C)**

# Violation of Assertions

If the assertion expression evaluates to false, it's an ERROR

- either in the algorithmic logic

-

- or in the implementation of an otherwise proved algorithm

# An Example

```
Insert (value: T)
   Before execution
              assert
                    1. count < capacity

     ... ... .Code for insert ... ...

   After execution
           assert
                1. count = old count+1
                2. count <= capacity
                3. values[old count]=value
```

# Assertions in Practice

**Proof view**

Assertions serve as specifications
(necessary and sufficient)

**Contract view**

Needs to be enforced by following it as a contract
A good design process (give and take)

**Defensive programming view**

An assertion expresses programmer’s intentions
Failure? – handle exception/abort
A good debugging process

# The C Assert Macro

[in  C++, use #include<cassert>]

```
#include <assert.h>
....
void insert (int i) {
        assert (count < CAPACITY);
        ... ..
}
main () {
        ...  insert (element); ...
}
```

# Types of Assertions

**Preconditions**

To be asserted before method execution begins

**Postconditions**

To be asserted after method execution before returning the result

**Class Invariants**

To be asserted

- after every object creation
- after every method execution
  - i.e. in observable states only, not necessarily during method execution

# Assertions vs. Exceptions

Exceptions are meant more for runtime handling of abnormalities
to provide fail-safe paths

when there are “recognized” abnormalities, or even for unexpected states resulting out of problems with the program

Assertions are often used to understand, to track development, and they may be turned off during runtime
or they could be taken care of by exception handling paths

```
#include <iostream>


#include <cassert>
using namespace std;

int main () {

 int n;
 cin >> n;

 int a[n];

 for (int i=0;i<=n; i++) {
     assert(i<n);
     assert(i>-1);

     a[i] =i;
 }
}
```

**An Example Program**

**The First Assertion Fails**

```
#include <iostream>
#define NDEBUG  //  turns off assertions

#include <cassert>
using namespace std;

int main () {

 int n;
 cin >> n;

 int a[n];

 for (int i=0;i<=n; i++) {
     assert(i<n);
     assert(i>-1);

     a[i] =i;
 }
}
```

**If NDEBUG is defined,**
**the assertions are turned off**
**i.e. they are not included in the shipment**