Dhirubhai Ambani Institute of Information & Communication Technology

IT314 - Software Engineering
Instructor - Prof. Saurabh Tiwari

Lab07: Program Inspection, Debugging and Static Analysis

Name: Tathya Prajapati
StudentId: 202201170

Q1) Program Inspection/Debugging for Long-code from Github

We are given the following checklist and we have to find all the possible errors accordingly,

1. Data referencing Errors
2. Data declaration Errors
3. Computation Errors
4. Comparison Errors
5. Control Flow errors
6. Interface errors
7. Input/Output Errors
8. Other Checks

First code is a Node JS file from

https://github.com/Medium/medium-sdk-nodejs/blob/master/test/mediumClient_test.js

The first half of the code is provided here ::

```
var medium = require("../")
var nock = require("nock")
var qs = require('querystring')
var should = require("should")
var url = require('url')


describe('MediumClient - constructor', function () {

  it('should throw a MediumError when options are undefined', function (done) {
    (function () { new medium.MediumClient() }).should.throw(medium.MediumError)
    done()
  })

  it('should throw a MediumError when options are empty', function (done) {
    (function () { new medium.MediumClient({}) }).should.throw(medium.MediumError)
    done()
  })

  it('should throw a MediumError when only clientId is provided', function (done) {
    (function () { new medium.MediumClient({clientId: 'xxx'})
}).should.throw(medium.MediumError)
    done()
  })

  it('should throw a MediumError when only clientSecret is provided', function (done) {
    (function () { new medium.MediumClient({clientSecret: 'yyy'})
}).should.throw(medium.MediumError)
    done()
  })

  it('should succeed when both clientId and clientSecret are provided', function (done) {
    var client = new medium.MediumClient({clientId: 'xxx', clientSecret: 'yyy'})
    done()
  })
})


describe('MediumClient - methods', function () {
```

```
  var clientId = 'xxx'
  var clientSecret = 'yyy'
  var client

  beforeEach(function () {
    client = new medium.MediumClient({clientId: clientId, clientSecret: clientSecret})
    nock.disableNetConnect()
  })

  afterEach(function () {
    nock.enableNetConnect();
    delete client
  })

  describe('#setAccessToken', function () {

    it ('sets the access token', function (done) {
      var token = "new token"
      client.setAccessToken(token)
      client._accessToken.should.be.String().and.equal(token)
      done()
    })
  })

  describe('#getAuthorizationUrl', function () {

    it ('returns a valid URL for fetching', function (done) {
      var state = "state"
      var redirectUrl = "https://example.com/callback"
      var scope = [medium.Scope.BASIC_PROFILE, medium.Scope.LIST_PUBLICATIONS,
medium.Scope.PUBLISH_POST]
      var authUrlStr = client.getAuthorizationUrl(state, redirectUrl, scope)
      var authUrl = url.parse(authUrlStr, true)
      authUrl.protocol.should.equal('https:')
      authUrl.hostname.should.equal('medium.com')
      authUrl.pathname.should.equal('/m/oauth/authorize')
      authUrl.query.should.deepEqual({
        client_id: clientId,
        scope: scope.join(','),
        response_type: 'code',
        state: state,
        redirect_uri: redirectUrl
      })
      done()
    })
  })
```

```
  describe('#exchangeAuthorizationCode', function () {

    it ('makes a request for authorization_code and sets the access token from response',
function (done) {
      var code = '12345'
      var grantType = 'authorization_code'
      var redirectUrl = 'https://example.com/callback'

      var requestBody = qs.stringify({
        code: code,
        client_id: clientId,
        client_secret: clientSecret,
        grant_type: grantType,
        redirect_uri: redirectUrl
      })
      // the response might have other parameters. this test only considers the ones called
out
      // in the Medium Node SDK documentation
      var accessToken = 'abcdef'
      var refreshToken = 'ghijkl'
      var responseBody = {
        access_token: accessToken,
        refresh_token: refreshToken
      }
      var request = nock('https://api.medium.com/', {
          'Content-Type': 'application/x-www-form-urlencoded'
        })
        .post('/v1/tokens', requestBody)
        .reply(201, responseBody)

      client.exchangeAuthorizationCode(code, redirectUrl, function (err, data) {
        if (err) throw err
        data.access_token.should.equal(accessToken)
        data.refresh_token.should.equal(refreshToken)
        done()
      })
      request.done()
    })
  })

  describe('#exchangeRefreshToken', function () {

    it ('makes a request for authorization_code and sets the access token from response',
function (done) {
      var refreshToken = 'fedcba'
      var accessToken = 'lkjihg'
```

```
      var requestBody = qs.stringify({
        refresh_token: refreshToken,
        client_id: clientId,
        client_secret: clientSecret,
        grant_type: 'refresh_token'
      })
      // the response might have other parameters. this test only considers the ones called
out
      // in the Medium Node SDK documentation
      var responseBody = {
        access_token: accessToken,
        refresh_token: refreshToken
      }
      var request = nock('https://api.medium.com/', {
          'Content-Type': 'application/x-www-form-urlencoded'
        })
        .post('/v1/tokens', requestBody)
        .reply(201, responseBody)

      client.exchangeRefreshToken(refreshToken, function (err, data) {
        if (err) throw err
        data.access_token.should.equal(accessToken)
        data.refresh_token.should.equal(refreshToken)
        done()
      })
      request.done()
    })
  })
```

1. Data Referencing Errors

- None found.

2. Data Declaration Errors

- None found.

3. Computation Errors

- None found.

4. Comparison Errors

- The assertion client._accessToken.should.be.String().and.equal(token) in the setAccessToken test is checking if _accessToken is a string before ensuring its equality. The order of assertions could lead to an unhandled error if _accessToken is not defined or is not a string.

5. Control Flow Errors

- None found.

6. Interface Errors

- The method client.setAccessToken(token) is called in the setAccessToken test, but if setAccessToken is not implemented correctly, it could lead to unexpected behavior.

7. Input/Output Errors

- The request.done() line in both exchangeAuthorizationCode and exchangeRefreshToken tests is incorrectly placed; it should be called after the request is executed, not directly after the request declaration. This may lead to premature invocation of done() in the context of network requests.

Next half of the code is here ::

```
  describe('#getUser', function () {
    it ('gets the information from expected URL and returns contents of data envelope',
function (done) {
      var response = { data: 'response data' }

      var request = nock('https://api.medium.com')
        .get('/v1/me')
        .reply(200, response)

      client.getUser(function (err, data) {
        if (err) throw err
        data.should.deepEqual(response['data'])
        done()
```

```
      })
      request.done()
    })
  })

  describe('#getPublicationsForUser', function () {

    it ('throws a MediumError when no user ID is provided', function (done) {
      (function () { client.getPublicationsForUser({}) }).should.throw(medium.MediumError)
      done()
    })

    it ('makes a proper GET request to the Medium API and returns contents of data envelope
when valid options are provided', function (done) {
      var userId = '123456'
      var response = { data: 'response data' }

      var request = nock('https://api.medium.com/')
        .get('/v1/users/' + userId + '/publications')
        .reply(200, response)

      client.getPublicationsForUser({userId: userId}, function (err, data) {
        if (err) throw err
        data.should.deepEqual(response['data'])
        done()
      })
      request.done()
    })
  })

  describe('#getContributorsForPublication', function () {

    it ('throws a MediumError when no publication ID is provided', function (done) {
      (function () { client.getContributorsForPublication({})
}).should.throw(medium.MediumError)
      done()
    })

    it ('makes a proper GET request to the Medium API and returns contents of data envelope',
function (done) {
      var options = { publicationId: 'abcdef' }
      var response = { data: 'response data' }
      var request = nock('https://api.medium.com/')
        .get('/v1/publications/' + options.publicationId + '/contributors')
        .reply(200, response)

      client.getContributorsForPublication(options, function (err, data) {
```

```
        if (err) throw err
        data.should.deepEqual(response['data'])
        done()
      })
      request.done()
    })
  })

  describe('#createPost', function () {

    it ('makes a proper POST request to the Medium API and returns contents of data envelope',
function (done) {
      var options = {
        userId: '123456',
        title: 'new post title',
        content: '<h1>New Post!</h1>',
        contentFormat: 'html',
        tags: ['js', 'unit tests'],
        canonicalUrl: 'http://example.com/new-post',
        publishedAt: '2004-02-12T15:19:21+00:00',
        publishStatus: 'draft',
        license: 'all-rights-reserved'
      }
      var response = { data: 'response data' }
      var request = nock('https://api.medium.com/')
        .post('/v1/users/' + options.userId + '/posts', {
            title: options.title,
            content: options.content,
            contentFormat: options.contentFormat,
            tags: options.tags,
            canonicalUrl: options.canonicalUrl,
            publishedAt: options.publishedAt,
            publishStatus: options.publishStatus,
            license: options.license
        })
        .reply(200, response)

      client.createPost(options, function (err, data) {
        if (err) throw err
        data.should.deepEqual(response['data'])
        done()
      })
      request.done()
    })
  })

  describe('#createPostInPublication', function () {
```

```
    it ('should throw an error when no publication ID is provided', function (done) {
      (function () { client.createPostInPublication({}) }).should.throw(medium.MediumError)
      done()
    })

    it ('makes a proper POST request to the Medium API and returns contents of data envelope',
function (done) {
      var options = {
        publicationId: 'abcdef',
        title: 'new post title',
        content: '<h1>New Post!</h1>',
        contentFormat: 'html',
        tags: ['js', 'unit tests'],
        canonicalUrl: 'http://example.com/new-post',
        publishedAt: '2004-02-12T15:19:21+00:00',
        publishStatus: 'draft',
        license: 'all-rights-reserved'
      }
      var response = { data: 'response data' }
      var request = nock('https://api.medium.com/')
        .post('/v1/publications/' + options.publicationId + '/posts', {
            title: options.title,
            content: options.content,
            contentFormat: options.contentFormat,
            tags: options.tags,
            canonicalUrl: options.canonicalUrl,
            publishedAt: options.publishedAt,
            publishStatus: options.publishStatus,
            license: options.license
        })
        .reply(200, response)

      client.createPostInPublication(options, function (err, data) {
        if (err) throw err
        data.should.deepEqual(response['data'])
        done()
      })
      request.done()
    })
  })
})
```

Here are the identified errors classified according to your categories in the provided code:

1. Data Referencing Errors
- None found.

2. Data Declaration Errors
- None found.

3. Computation Errors
- None found.

4. Comparison Errors
- In the tests for getUser, getPublicationsForUser, getContributorsForPublication, createPost, and createPostInPublication, the assertion data.should.deepEqual(response['data']) assumes that response['data'] contains the expected output structure. If response.data is not defined correctly or is different in structure, this could lead to a comparison error.

5. Control Flow Errors
- None found.

6. Interface Errors
- None found.

7. Input/Output Errors
- The request.done() call in each test should be executed after the request is processed (after the callback), rather than immediately after the request declaration. This could lead to improper handling of request expectations.

2nd Code is a sample of small Operating system

https://github.com/nuta/operating-system-in-1000-lines/tree/main

There are sub parts to it so I placed all of them in random order...

```
#include "common.h"

void *memset(void *buf, char c, size_t n) {
    uint8_t *p = (uint8_t *) buf;
    while (n--)
        *p++ = c;
    return buf;
}

void *memcpy(void *dst, const void *src, size_t n) {
    uint8_t *d = (uint8_t *) dst;
    const uint8_t *s = (const uint8_t *) src;
    while (n--)
        *d++ = *s++;
    return dst;
}

char *strcpy(char *dst, const char *src) {
    char *d = dst;
    while (*src)
        *d++ = *src++;
    *d = '\0';
    return dst;
}

int strcmp(const char *s1, const char *s2) {
    while (*s1 && *s2) {
        if (*s1 != *s2)
            break;
```

```
        s1++;
        s2++;
    }

    return *(unsigned char *)s1 - *(unsigned char *)s2;
}

void putchar(char ch);

void printf(const char *fmt, ...) {
    va_list vargs;
    va_start(vargs, fmt);
    while (*fmt) {
        if (*fmt == '%') {
            fmt++;
            switch (*fmt) {
                case '\0':
                    putchar('%');
                    goto end;
                case '%':
                    putchar('%');
                    break;
                case 's': {
                    const char *s = va_arg(vargs, const char *);
                    while (*s) {
                        putchar(*s);
                        s++;
                    }
                    break;
                }
                case 'd': {
                    int value = va_arg(vargs, int);
                    if (value < 0) {
                        putchar('-');
                        value = -value;
```

```
                    }

                    int divisor = 1;
                    while (value / divisor > 9)
                        divisor *= 10;

                    while (divisor > 0) {
                        putchar('0' + value / divisor);
                        value %= divisor;
                        divisor /= 10;
                    }

                    break;
                }
                case 'x': {
                    int value = va_arg(vargs, int);
                    for (int i = 7; i >= 0; i--) {
                        int nibble = (value >> (i * 4)) & 0xf;
                        putchar("0123456789abcdef"[nibble]);
                    }
                }
            }
        } else {
            putchar(*fmt);
        }

        fmt++;
    }

end:
    va_end(vargs);
}
```

1. Data Referencing Errors

- None found.

2. Data Declaration Errors

- In the printf function, the va_list vargs is declared but not properly handled. If va_end(vargs) is called without a corresponding va_start(vargs, fmt), it could lead to undefined behavior, although this isn't directly indicated here since va_start is correctly used before va_end.

3. Computation Errors

- None found.

4. Comparison Errors

- None found.

5. Control Flow Errors

- In the printf function, the goto end; statement inside the switch block can create confusion. Although it is not an error, using goto can lead to less readable code and should be avoided if possible.

6. Interface Errors

- The putchar function is declared but not defined in the provided code. This could lead to linker errors if putchar is called without a definition available.
- The function printf uses various formats (%d, %x, %s), but there is no error handling for unsupported formats, which could lead to unpredictable behavior if an unsupported format specifier is encountered.

7. Input/Output Errors

- In the printf function, there is no check for a null pointer in the const char *s = va_arg(vargs, const char *); line for the string format specifier (%s). If a null pointer is passed, it could lead to dereferencing a null pointer and cause a segmentation fault.

```
#include "kernel.h"
#include "common.h"

extern char __kernel_base[];
extern char __stack_top[];
extern char __bss[], __bss_end[];
extern char __free_ram[], __free_ram_end[];
extern char _binary_shell_bin_start[], _binary_shell_bin_size[];

struct process procs[PROCS_MAX];
struct process *current_proc;
struct process *idle_proc;

paddr_t alloc_pages(uint32_t n) {
    static paddr_t next_paddr = (paddr_t) __free_ram;
    paddr_t paddr = next_paddr;
    next_paddr += n * PAGE_SIZE;

    if (next_paddr > (paddr_t) __free_ram_end)
        PANIC("out of memory");

    memset((void *) paddr, 0, n * PAGE_SIZE);
    return paddr;
}

void map_page(uint32_t *table1, uint32_t vaddr, paddr_t paddr, uint32_t flags) {
    if (!is_aligned(vaddr, PAGE_SIZE))
        PANIC("unaligned vaddr %x", vaddr);

    if (!is_aligned(paddr, PAGE_SIZE))
        PANIC("unaligned paddr %x", paddr);

    uint32_t vpn1 = (vaddr >> 22) & 0x3ff;
    if ((table1[vpn1] & PAGE_V) == 0) {
        uint32_t pt_paddr = alloc_pages(1);
        table1[vpn1] = ((pt_paddr / PAGE_SIZE) << 10) | PAGE_V;
    }
```

```
    uint32_t vpn0 = (vaddr >> 12) & 0x3ff;
    uint32_t *table0 = (uint32_t *) ((table1[vpn1] >> 10) * PAGE_SIZE);
    table0[vpn0] = ((paddr / PAGE_SIZE) << 10) | flags | PAGE_V;
}

struct sbiret sbi_call(long arg0, long arg1, long arg2, long arg3, long arg4,
                       long arg5, long fid, long eid) {
    register long a0 __asm__("a0") = arg0;
    register long a1 __asm__("a1") = arg1;
    register long a2 __asm__("a2") = arg2;
    register long a3 __asm__("a3") = arg3;
    register long a4 __asm__("a4") = arg4;
    register long a5 __asm__("a5") = arg5;
    register long a6 __asm__("a6") = fid;
    register long a7 __asm__("a7") = eid;

    __asm__ __volatile__("ecall"
                         : "=r"(a0), "=r"(a1)
                         : "r"(a0), "r"(a1), "r"(a2), "r"(a3), "r"(a4), "r"(a5),
                           "r"(a6), "r"(a7)
                         : "memory");
    return (struct sbiret){.error = a0, .value = a1};
}

struct virtio_virtq *blk_request_vq;
struct virtio_blk_req *blk_req;
paddr_t blk_req_paddr;
unsigned blk_capacity;

uint32_t virtio_reg_read32(unsigned offset) {
    return *((volatile uint32_t *) (VIRTIO_BLK_PADDR + offset));
}

uint64_t virtio_reg_read64(unsigned offset) {
    return *((volatile uint64_t *) (VIRTIO_BLK_PADDR + offset));
}

void virtio_reg_write32(unsigned offset, uint32_t value) {
    *((volatile uint32_t *) (VIRTIO_BLK_PADDR + offset)) = value;
}

void virtio_reg_fetch_and_or32(unsigned offset, uint32_t value) {
    virtio_reg_write32(offset, virtio_reg_read32(offset) | value);
}

bool virtq_is_busy(struct virtio_virtq *vq) {
    return vq->last_used_index != *vq->used_index;
```

```
}

void virtq_kick(struct virtio_virtq *vq, int desc_index) {
    vq->avail.ring[vq->avail.index % VIRTQ_ENTRY_NUM] = desc_index;
    vq->avail.index++;
    __sync_synchronize();
    virtio_reg_write32(VIRTIO_REG_QUEUE_NOTIFY, vq->queue_index);
    vq->last_used_index++;
}

struct virtio_virtq *virtq_init(unsigned index) {
    paddr_t virtq_paddr = alloc_pages(align_up(sizeof(struct virtio_virtq), PAGE_SIZE) /
PAGE_SIZE);
    struct virtio_virtq *vq = (struct virtio_virtq *) virtq_paddr;
    vq->queue_index = index;
    vq->used_index = (volatile uint16_t *) &vq->used.index;
    virtio_reg_write32(VIRTIO_REG_QUEUE_SEL, index);
    virtio_reg_write32(VIRTIO_REG_QUEUE_NUM, VIRTQ_ENTRY_NUM);
    virtio_reg_write32(VIRTIO_REG_QUEUE_ALIGN, 0);
    virtio_reg_write32(VIRTIO_REG_QUEUE_PFN, virtq_paddr);
    return vq;
}

void virtio_blk_init(void) {
    if (virtio_reg_read32(VIRTIO_REG_MAGIC) != 0x74726976)
        PANIC("virtio: invalid magic value");
    if (virtio_reg_read32(VIRTIO_REG_VERSION) != 1)
        PANIC("virtio: invalid version");
    if (virtio_reg_read32(VIRTIO_REG_DEVICE_ID) != VIRTIO_DEVICE_BLK)
        PANIC("virtio: invalid device id");

    virtio_reg_write32(VIRTIO_REG_DEVICE_STATUS, 0);
    virtio_reg_fetch_and_or32(VIRTIO_REG_DEVICE_STATUS, VIRTIO_STATUS_ACK);
    virtio_reg_fetch_and_or32(VIRTIO_REG_DEVICE_STATUS, VIRTIO_STATUS_DRIVER);
    virtio_reg_fetch_and_or32(VIRTIO_REG_DEVICE_STATUS, VIRTIO_STATUS_FEAT_OK);
    blk_request_vq = virtq_init(0);
    virtio_reg_write32(VIRTIO_REG_DEVICE_STATUS, VIRTIO_STATUS_DRIVER_OK);

    blk_capacity = virtio_reg_read64(VIRTIO_REG_DEVICE_CONFIG + 0) * SECTOR_SIZE;
    printf("virtio-blk: capacity is %d bytes\n", blk_capacity);

    blk_req_paddr = alloc_pages(align_up(sizeof(*blk_req), PAGE_SIZE) / PAGE_SIZE);
    blk_req = (struct virtio_blk_req *) blk_req_paddr;
}
```

1. Data Referencing Errors

- None identified.

2. Data Declaration Errors

- None identified.

3. Computation Errors

- None identified.

4. Comparison Errors

- None identified.

5. Control Flow Errors

- No check for successful allocation in virtq_init() after alloc_pages(). This could lead to dereferencing a NULL pointer.

6. Interface Errors

- No explicit validation for register offsets in virtio_reg_read32, virtio_reg_read64, and related functions.

7. Input/Output Errors

- None identified.

```
void read_write_disk(void *buf, unsigned sector, int is_write) {
    if (sector >= blk_capacity / SECTOR_SIZE) {
        printf("virtio: tried to read/write sector=%d, but capacity is %d\n",
               sector, blk_capacity / SECTOR_SIZE);
        return;
    }

    blk_req->sector = sector;
    blk_req->type = is_write ? VIRTIO_BLK_T_OUT : VIRTIO_BLK_T_IN;

    if (is_write)
        memcpy(blk_req->data, buf, SECTOR_SIZE);
```

```
    struct virtio_virtq *vq = blk_request_vq;
    vq->descs[0].addr = blk_req_paddr;
    vq->descs[0].len = sizeof(uint32_t) * 2 + sizeof(uint64_t);
    vq->descs[0].flags = VIRTQ_DESC_F_NEXT;
    vq->descs[0].next = 1;

    vq->descs[1].addr = blk_req_paddr + offsetof(struct virtio_blk_req, data);
    vq->descs[1].len = SECTOR_SIZE;
    vq->descs[1].flags = VIRTQ_DESC_F_NEXT | (is_write ? 0 : VIRTQ_DESC_F_WRITE);
    vq->descs[1].next = 2;

    vq->descs[2].addr = blk_req_paddr + offsetof(struct virtio_blk_req, status);
    vq->descs[2].len = sizeof(uint8_t);
    vq->descs[2].flags = VIRTQ_DESC_F_WRITE;

    virtq_kick(vq, 0);
    while (virtq_is_busy(vq))
        ;

    if (blk_req->status != 0) {
        printf("virtio: warn: failed to read/write sector=%d status=%d\n",
               sector, blk_req->status);
        return;
    }

    if (!is_write)
        memcpy(buf, blk_req->data, SECTOR_SIZE);
}

struct file files[FILES_MAX];
uint8_t disk[DISK_MAX_SIZE];

int oct2int(char *oct, int len) {
    int dec = 0;
    for (int i = 0; i < len; i++) {
        if (oct[i] < '0' || oct[i] > '7')
            break;

        dec = dec * 8 + (oct[i] - '0');
    }
    return dec;
}

void fs_flush(void) {
    memset(disk, 0, sizeof(disk));
    unsigned off = 0;
```

```
    for (int file_i = 0; file_i < FILES_MAX; file_i++) {
        struct file *file = &files[file_i];
        if (!file->in_use)
            continue;

        struct tar_header *header = (struct tar_header *) &disk[off];
        memset(header, 0, sizeof(*header));
        strcpy(header->name, file->name);
        strcpy(header->mode, "000644");
        strcpy(header->magic, "ustar");
        strcpy(header->version, "00");
        header->type = '0';

        int filesz = file->size;
        for (int i = sizeof(header->size); i > 0; i--) {
            header->size[i - 1] = (filesz % 8) + '0';
            filesz /= 8;
        }

        int checksum = ' ' * sizeof(header->checksum);
        for (unsigned i = 0; i < sizeof(struct tar_header); i++)
            checksum += (unsigned char) disk[off + i];

        for (int i = 5; i >= 0; i--) {
            header->checksum[i] = (checksum % 8) + '0';
            checksum /= 8;
        }

        memcpy(header->data, file->data, file->size);
        off += align_up(sizeof(struct tar_header) + file->size, SECTOR_SIZE);
    }

    for (unsigned sector = 0; sector < sizeof(disk) / SECTOR_SIZE; sector++)
        read_write_disk(&disk[sector * SECTOR_SIZE], sector, true);

    printf("wrote %d bytes to disk\n", sizeof(disk));
}

void fs_init(void) {
    for (unsigned sector = 0; sector < sizeof(disk) / SECTOR_SIZE; sector++)
        read_write_disk(&disk[sector * SECTOR_SIZE], sector, false);

    unsigned off = 0;
    for (int i = 0; i < FILES_MAX; i++) {
        struct tar_header *header = (struct tar_header *) &disk[off];
        if (header->name[0] == '\0')
            break;
```

```
        if (strcmp(header->magic, "ustar") != 0)
            PANIC("invalid tar header: magic=\"%s\"", header->magic);

        int filesz = oct2int(header->size, sizeof(header->size));
        struct file *file = &files[i];
        file->in_use = true;
        strcpy(file->name, header->name);
        memcpy(file->data, header->data, filesz);
        file->size = filesz;
        printf("file: %s, size=%d\n", file->name, file->size);

        off += align_up(sizeof(struct tar_header) + filesz, SECTOR_SIZE);
    }
}
```

1. Data Referencing Errors

- The code references blk_req, blk_capacity, blk_request_vq, and blk_req_paddr without showing their definitions. Make sure these variables are properly initialized and referenced.

2. Data Declaration Errors

- The variable disk is declared with uint8_t disk[DISK_MAX_SIZE];, but there's no indication of the value assigned to DISK_MAX_SIZE. Ensure it's defined somewhere.
- The struct tar_header is referenced without a declaration in the provided code. Ensure it is defined correctly in your project.

3. Computation Errors

- The calculation of filesz in fs_flush does not account for the potential overflow when calculating the checksum. Although the tar format specifies a maximum size, it's a good practice to check sizes to avoid overflow.

- In the oct2int function, if the input oct string has more than three characters (which represent a valid octal digit), the conversion might give unexpected results. Consider adding a limit on len.

4. Comparison Errors
- In fs_init, the check if (strcmp(header->magic, "ustar") != 0) is valid, but the code doesn't handle the case where header->magic could be NULL. Consider adding a NULL check before comparison.

5. Control Flow Errors
- The read_write_disk function might enter an infinite loop if the disk request is never completed. Ensure that virtq_kick(vq, 0) and virtq_is_busy(vq) are implemented correctly to handle this situation.
- The function fs_flush will print that it has written to the disk regardless of whether the write was successful. Consider checking for errors in read_write_disk.

6. Interface Errors
- The putchar function must be defined elsewhere, or else there will be linking errors when compiling.
- Ensure that align_up is properly defined and that its purpose is clear; it seems to be intended for aligning data sizes, but its implementation is not provided here.

7. Input/Output Errors
- In the fs_flush function, when writing to the disk, if read_write_disk fails for any reason (e.g., due to a full disk or hardware failure), the user is not notified. Implement error handling to manage this.

- In the fs_init function, if the data read from the disk doesn't match the expected format or the file size exceeds DISK_MAX_SIZE, it may cause out-of-bounds memory access when populating the file structures.

```
struct file *fs_lookup(const char *filename) {
    for (int i = 0; i < FILES_MAX; i++) {
        struct file *file = &files[i];
        if (!strcmp(file->name, filename))
            return file;
    }

    return NULL;
}

void putchar(char ch) {
    sbi_call(ch, 0, 0, 0, 0, 0, 0, 1 /* Console Putchar */);
}

long getchar(void) {
    struct sbiret ret = sbi_call(0, 0, 0, 0, 0, 0, 0, 2);
    return ret.error;
}

__attribute__((naked))
__attribute__((aligned(4)))
void kernel_entry(void) {
    __asm__ __volatile__(
        "csrrw sp, sscratch, sp\n"
        "addi sp, sp, -4 * 31\n"
        "sw ra,  4 * 0(sp)\n"
        "sw gp,  4 * 1(sp)\n"
        "sw tp,  4 * 2(sp)\n"
        "sw t0,  4 * 3(sp)\n"
        "sw t1,  4 * 4(sp)\n"
        "sw t2,  4 * 5(sp)\n"
        "sw t3,  4 * 6(sp)\n"
        "sw t4,  4 * 7(sp)\n"
        "sw t5,  4 * 8(sp)\n"
        "sw t6,  4 * 9(sp)\n"
        "sw a0,  4 * 10(sp)\n"
        "sw a1,  4 * 11(sp)\n"
        "sw a2,  4 * 12(sp)\n"
        "sw a3,  4 * 13(sp)\n"
        "sw a4,  4 * 14(sp)\n"
        "sw a5,  4 * 15(sp)\n"
```

```
        "sw a6,  4 * 16(sp)\n"
        "sw a7,  4 * 17(sp)\n"
        "sw s0,  4 * 18(sp)\n"
        "sw s1,  4 * 19(sp)\n"
        "sw s2,  4 * 20(sp)\n"
        "sw s3,  4 * 21(sp)\n"
        "sw s4,  4 * 22(sp)\n"
        "sw s5,  4 * 23(sp)\n"
        "sw s6,  4 * 24(sp)\n"
        "sw s7,  4 * 25(sp)\n"
        "sw s8,  4 * 26(sp)\n"
        "sw s9,  4 * 27(sp)\n"
        "sw s10, 4 * 28(sp)\n"
        "sw s11, 4 * 29(sp)\n"

        "csrr a0, sscratch\n"
        "sw a0,  4 * 30(sp)\n"

        "addi a0, sp, 4 * 31\n"
        "csrw sscratch, a0\n"

        "mv a0, sp\n"
        "call handle_trap\n"

        "lw ra,  4 * 0(sp)\n"
        "lw gp,  4 * 1(sp)\n"
        "lw tp,  4 * 2(sp)\n"
        "lw t0,  4 * 3(sp)\n"
        "lw t1,  4 * 4(sp)\n"
        "lw t2,  4 * 5(sp)\n"
        "lw t3,  4 * 6(sp)\n"
        "lw t4,  4 * 7(sp)\n"
        "lw t5,  4 * 8(sp)\n"
        "lw t6,  4 * 9(sp)\n"
        "lw a0,  4 * 10(sp)\n"
        "lw a1,  4 * 11(sp)\n"
        "lw a2,  4 * 12(sp)\n"
        "lw a3,  4 * 13(sp)\n"
        "lw a4,  4 * 14(sp)\n"
        "lw a5,  4 * 15(sp)\n"
        "lw a6,  4 * 16(sp)\n"
        "lw a7,  4 * 17(sp)\n"
        "lw s0,  4 * 18(sp)\n"
        "lw s1,  4 * 19(sp)\n"
        "lw s2,  4 * 20(sp)\n"
        "lw s3,  4 * 21(sp)\n"
        "lw s4,  4 * 22(sp)\n"
```

```
        "lw s5,  4 * 23(sp)\n"
        "lw s6,  4 * 24(sp)\n"
        "lw s7,  4 * 25(sp)\n"
        "lw s8,  4 * 26(sp)\n"
        "lw s9,  4 * 27(sp)\n"
        "lw s10, 4 * 28(sp)\n"
        "lw s11, 4 * 29(sp)\n"
        "lw sp,  4 * 30(sp)\n"
        "sret\n"
    );
}

__attribute__((naked)) void user_entry(void) {
    __asm__ __volatile__(
        "csrw sepc, %[sepc]\n"
        "csrw sstatus, %[sstatus]\n"
        "sret\n"
        :
        : [sepc] "r" (USER_BASE),
          [sstatus] "r" (SSTATUS_SPIE | SSTATUS_SUM)
    );
}

__attribute__((naked)) void switch_context(uint32_t *prev_sp,
                                           uint32_t *next_sp) {
    __asm__ __volatile__(
        "addi sp, sp, -13 * 4\n"
        "sw ra,  0  * 4(sp)\n"
        "sw s0,  1  * 4(sp)\n"
        "sw s1,  2  * 4(sp)\n"
        "sw s2,  3  * 4(sp)\n"
        "sw s3,  4  * 4(sp)\n"
        "sw s4,  5  * 4(sp)\n"
        "sw s5,  6  * 4(sp)\n"
        "sw s6,  7  * 4(sp)\n"
        "sw s7,  8  * 4(sp)\n"
        "sw s8,  9  * 4(sp)\n"
        "sw s9,  10 * 4(sp)\n"
        "sw s10, 11 * 4(sp)\n"
        "sw s11, 12 * 4(sp)\n"
        "sw sp, (a0)\n"
        "lw sp, (a1)\n"
        "lw ra,  0  * 4(sp)\n"
        "lw s0,  1  * 4(sp)\n"
        "lw s1,  2  * 4(sp)\n"
        "lw s2,  3  * 4(sp)\n"
        "lw s3,  4  * 4(sp)\n"
```

```
        "lw s4,  5  * 4(sp)\n"
        "lw s5,  6  * 4(sp)\n"
        "lw s6,  7  * 4(sp)\n"
        "lw s7,  8  * 4(sp)\n"
        "lw s8,  9  * 4(sp)\n"
        "lw s9,  10 * 4(sp)\n"
        "lw s10, 11 * 4(sp)\n"
        "lw s11, 12 * 4(sp)\n"
        "addi sp, sp, 13 * 4\n"
        "ret\n"
    );
}

struct process *create_process(const void *image, size_t image_size) {
    struct process *proc = NULL;
    int i;
    for (i = 0; i < PROCS_MAX; i++) {
        if (procs[i].state == PROC_UNUSED) {
            proc = &procs[i];
            break;
        }
    }

    if (!proc)
        PANIC("no free process slots");

    uint32_t *sp = (uint32_t *) &proc->stack[sizeof(proc->stack)];
    *--sp = 0;                      // s11
    *--sp = 0;                      // s10
    *--sp = 0;                      // s9
    *--sp = 0;                      // s8
    *--sp = 0;                      // s7
    *--sp = 0;                      // s6
    *--sp = 0;                      // s5
    *--sp = 0;                      // s4
    *--sp = 0;                      // s3
    *--sp = 0;                      // s2
    *--sp = 0;                      // s1
    *--sp = 0;                      // s0
    *--sp = (uint32_t) user_entry;  // ra

    uint32_t *page_table = (uint32_t *) alloc_pages(1);

    // Kernel pages.
    for (paddr_t paddr = (paddr_t) __kernel_base;
         paddr < (paddr_t) __free_ram_end; paddr += PAGE_SIZE)
        map_page(page_table, paddr, paddr, PAGE_R | PAGE_W | PAGE_X);
```

```
    // virtio-blk
    map_page(page_table, VIRTIO_BLK_PADDR, VIRTIO_BLK_PADDR, PAGE_R | PAGE_W);

    // User pages.
    for (uint32_t off = 0; off < image_size; off += PAGE_SIZE) {
        paddr_t page = alloc_pages(1);
        memcpy((void *) page, image + off, PAGE_SIZE);
        map_page(page_table, USER_BASE + off, page,
                 PAGE_U | PAGE_R | PAGE_W | PAGE_X);
    }

    proc->pid = i + 1;
    proc->state = PROC_RUNNABLE;
    proc->sp = (uint32_t) sp;
    proc->page_table = page_table;
    return proc;
}
```

1. Data Referencing Errors

- Potential Null Pointer Dereference: The fs_lookup function assumes that files is initialized and valid. If files is uninitialized or if FILES_MAX is set to 0, it may lead to undefined behavior.

2. Data Declaration Errors

- Missing Struct Definition: The struct file and the files array are referenced but not defined in the provided code. This could lead to compilation errors if they are not declared elsewhere in the program.

3. Computation Errors

- None found.

4. Comparison Errors

- None found.

5. Control Flow Errors

- Unconditional Exit: The PANIC("no free process slots"); call does not handle the case where proc is NULL gracefully, potentially leading to abrupt termination of the program. Instead, it should ideally return or clean up resources.

6. Interface Errors

- None found.

7. Input/Output Errors

- Buffer Overrun Risk: The loop that initializes the stack (with *--sp = 0;) assumes that the stack has sufficient space. If the size of proc->stack is less than expected, it may result in a stack overflow.

```
void yield(void) {
    struct process *next = idle_proc;
    for (int i = 0; i < PROCS_MAX; i++) {
        struct process *proc = &procs[(current_proc->pid + i) % PROCS_MAX];
        if (proc->state == PROC_RUNNABLE && proc->pid > 0) {
            next = proc;
            break;
        }
    }

    if (next == current_proc)
        return;

    struct process *prev = current_proc;
    current_proc = next;

    __asm__ __volatile__(
        "sfence.vma\n"
        "csrw satp, %[satp]\n"
        "sfence.vma\n"
        "csrw sscratch, %[sscratch]\n"
        :
        : [satp] "r" (SATP_SV32 | ((uint32_t) next->page_table / PAGE_SIZE)),
          [sscratch] "r" ((uint32_t) &next->stack[sizeof(next->stack)])
    );

    switch_context(&prev->sp, &next->sp);
}
```

```c
void handle_syscall(struct trap_frame *f) {
    switch (f->a3) {
        case SYS_PUTCHAR:
            putchar(f->a0);
            break;
        case SYS_GETCHAR:
            while (1) {
                long ch = getchar();
                if (ch >= 0) {
                    f->a0 = ch;
                    break;
                }

                yield();
            }
            break;
        case SYS_EXIT:
            printf("process %d exited\n", current_proc->pid);
            current_proc->state = PROC_EXITED;
            yield();
            PANIC("unreachable");
        case SYS_READFILE:
        case SYS_WRITEFILE: {
            const char *filename = (const char *) f->a0;
            char *buf = (char *) f->a1;
            int len = f->a2;
            struct file *file = fs_lookup(filename);
            if (!file) {
                printf("file not found: %s\n", filename);
                f->a0 = -1;
                break;
            }

            if (len > (int) sizeof(file->data))
                len = file->size;

            if (f->a3 == SYS_WRITEFILE) {
                memcpy(file->data, buf, len);
                file->size = len;
                fs_flush();
            } else {
                memcpy(buf, file->data, len);
            }

            f->a0 = len;
            break;
```

```
        }
        default:
            PANIC("unexpected syscall a3=%x\n", f->a3);
    }
}

void handle_trap(struct trap_frame *f) {
    uint32_t scause = READ_CSR(scause);
    uint32_t stval = READ_CSR(stval);
    uint32_t user_pc = READ_CSR(sepc);
    if (scause == SCAUSE_ECALL) {
        handle_syscall(f);
        user_pc += 4;
    } else {
        PANIC("unexpected trap scause=%x, stval=%x, sepc=%x\n", scause, stval, user_pc);
    }

    WRITE_CSR(sepc, user_pc);
}

void kernel_main(void) {
    memset(__bss, 0, (size_t) __bss_end - (size_t) __bss);
    printf("\n\n");
    WRITE_CSR(stvec, (uint32_t) kernel_entry);
    virtio_blk_init();
    fs_init();

    idle_proc = create_process(NULL, 0);
    idle_proc->pid = -1; // idle
    current_proc = idle_proc;

    create_process(_binary_shell_bin_start, (size_t) _binary_shell_bin_size);
    yield();

    PANIC("switched to idle process");
}

__attribute__((section(".text.boot")))
__attribute__((naked))
void boot(void) {
    __asm__ __volatile__(
        "mv sp, %[stack_top]\n"
        "j kernel_main\n"
        :
        : [stack_top] "r" (__stack_top)
    );
}
```

Data Referencing Errors

- Potential Null Pointer Dereference: current_proc could be null if no processes have been created or if it has been improperly initialized before yield() is called.

Data Declaration Errors

- Uninitialized Variables: Variables such as idle_proc and current_proc may be used without proper initialization if create_process fails or if there are no processes.

Computation Errors

- Improper Memory Access: The calculation of next->page_table / PAGE_SIZE could lead to incorrect values if next->page_table is not properly aligned or initialized.

Comparison Errors

- Unsigned vs. Signed Comparison: Comparing proc->pid > 0 may cause unintended behavior if proc->pid is an unsigned type.

Control Flow Errors

- Infinite Loop Risk: The while (1) loop in handle_syscall for SYS_GETCHAR may lead to an infinite loop if getchar() never returns a valid character.

Interface Errors

- Missing Error Handling for System Calls: Functions like fs_lookup, memcpy, and printf may fail silently without error checking or reporting in certain scenarios.

Input/Output Errors

- Data Overwrite Risk: In handle_syscall for SYS_WRITEFILE, if len is not properly validated, it may lead to writing beyond the bounds of file->data.

```
#include "user.h"

void main(void) {
    while (1) {
prompt:
        printf("> ");
        char cmdline[128];
        for (int i = 0;; i++) {
            char ch = getchar();
            putchar(ch);
            if (i == sizeof(cmdline) - 1) {
                printf("command line too long\n");
                goto prompt;
            } else if (ch == '\r') {
                printf("\n");
                cmdline[i] = '\0';
                break;
            } else {
                cmdline[i] = ch;
            }
        }

        if (strcmp(cmdline, "hello") == 0)
            printf("Hello world from shell!\n");
        else if (strcmp(cmdline, "exit") == 0)
            exit();
        else if (strcmp(cmdline, "readfile") == 0) {
            char buf[128];
            int len = readfile("hello.txt", buf, sizeof(buf));
            buf[len] = '\0';
            printf("%s\n", buf);
        }
        else if (strcmp(cmdline, "writefile") == 0)
            writefile("hello.txt", "Hello from shell!\n", 19);
        else
            printf("unknown command: %s\n", cmdline);
    }
}
```

Data Referencing Errors
- Potential Buffer Overflow: The cmdline buffer is not properly null-terminated if the user inputs more than 127 characters (since one byte is used for the null terminator).

Data Declaration Errors
- Uninitialized Variable: The variable buf in the readfile command could be uninitialized if the file reading fails before it is populated.

Computation Errors
- Length Calculation: In the readfile command, the length returned by readfile() is used directly without checking if it exceeds the size of buf. If len is larger than 128, this could lead to a buffer overflow when setting buf[len] = '\0';.

Comparison Errors
- Use of strcmp: If cmdline is not properly null-terminated due to buffer overflow or a missed termination case, the behavior of strcmp can be undefined.

Control Flow Errors
- Infinite Loop Risk: The while (1) loop will run indefinitely unless a command that calls exit() is executed. There's no condition to break out of the loop except for exit().

Interface Errors
- Missing Error Handling: The return value of readfile is not checked for errors. If the file does not exist or read fails, it could lead to undefined behavior.

Input/Output Errors

- Data Overwrite Risk: In the writefile command, there is no check to ensure that the data being written is less than or equal to the length of the file buffer on the file system.

```
#include "user.h"

extern char __stack_top[];

int syscall(int sysno, int arg0, int arg1, int arg2) {
    register int a0 __asm__("a0") = arg0;
    register int a1 __asm__("a1") = arg1;
    register int a2 __asm__("a2") = arg2;
    register int a3 __asm__("a3") = sysno;

    __asm__ __volatile__("ecall"
                         : "=r"(a0)
                         : "r"(a0), "r"(a1), "r"(a2), "r"(a3)
                         : "memory");

    return a0;
}

void putchar(char ch) {
    syscall(SYS_PUTCHAR, ch, 0, 0);
}

int getchar(void) {
    return syscall(SYS_GETCHAR, 0, 0, 0);
}

int readfile(const char *filename, char *buf, int len) {
    return syscall(SYS_READFILE, (int) filename, (int) buf, len);
}

int writefile(const char *filename, const char *buf, int len) {
```

```
    return syscall(SYS_WRITEFILE, (int) filename, (int) buf, len);
}

__attribute__((noreturn)) void exit(void) {
    syscall(SYS_EXIT, 0, 0, 0);
    for (;;);
}

__attribute__((section(".text.start")))
__attribute__((naked))
void start(void) {
    __asm__ __volatile__(
        "mv sp, %[stack_top]\n"
        "call main\n"
        "call exit\n" ::[stack_top] "r"(__stack_top));
}
```

Data Referencing Errors

- Casting Pointers to Integers: The code casts const char *filename and char *buf to int, which can lead to data loss or corruption on architectures where pointers are larger than integers (e.g., 64-bit systems).

Data Declaration Errors

- Uninitialized Variables: If syscall fails or returns an error value, the variables buf and filename may not be handled properly in readfile and writefile functions, which could lead to unexpected behavior.

Computation Errors

- Return Value Ignored: In readfile and writefile, the return value from syscall is not checked. If the syscall fails (e.g., file not found), this could lead to undefined behavior when using the data later.

Comparison Errors

- No apparent comparison errors exist in the provided code.

Control Flow Errors

- Endless Loop in exit: The for (;;); loop in the exit function will create an infinite loop after the syscall call, which could indicate a lack of proper termination or error handling.

Interface Errors

- No Error Handling for System Calls: There is no error checking for the return values of syscall in any function. For instance, if a file operation fails, the error is not handled.

Input/Output Errors

- Invalid Memory Access: If buf in readfile or writefile points to an invalid or unallocated memory address, the code will attempt to read from or write to that memory location, leading to potential crashes or data corruption.

Q2) CODE DEBUGGING: Debugging is the process of localizing, analyzing, and removing suspected

Program-1: Armstrong.java

Code Debugging Analysis:

1. **Errors in the Code:** There were issues in extracting the digits and updating the number during the Armstrong number check. The errors involved incorrect operations for extracting the last digit and reducing the number, as mentioned below.

2. **Breakpoints Required:**
    - **Breakpoint 1:** Place a breakpoint at remainder = num % 10; to confirm that the last digit of the number is being extracted correctly.
    - **Breakpoint 2:** Place a breakpoint at num = num / 10; to check if the reduction of the number by removing the last digit is functioning as expected.
    - **Breakpoint 3:** Place a breakpoint at the if statement to ensure that the condition check == n is being evaluated correctly after all digits have been processed.

(A). **Steps Taken to Fix the Errors:**

- Modified remainder = num / 10; to remainder = num % 10; to ensure correct extraction of the last digit.
- Adjusted num = num % 10; to num = num / 10; to correctly remove the last digit from the number.

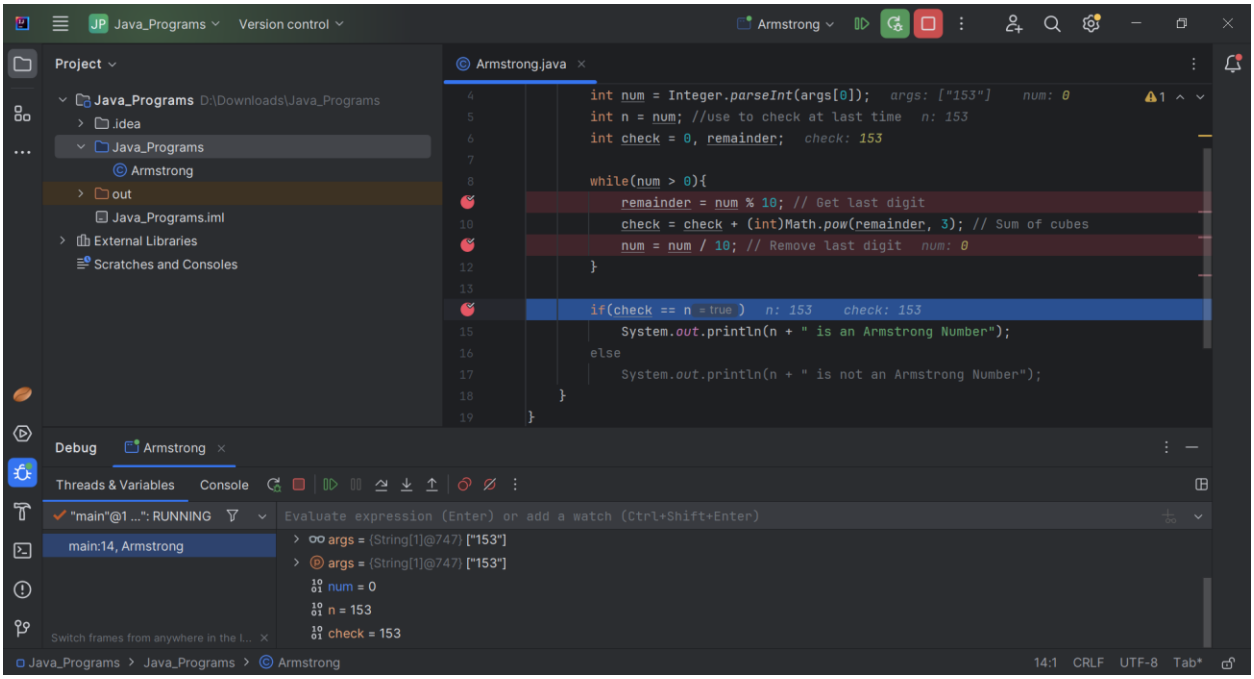
Debug without correction:

3. Complete Executable Code:

```
// Armstrong Number
class Armstrong{
    public static void main(String args[]){
        int num = Integer.parseInt(args[0]);
        int n = num; // use to check at last time
        int check = 0, remainder;

        while(num > 0){
            remainder = num % 10;
            check = check + (int)Math.pow(remainder, 3);
            num = num / 10;
        }
        if(check == n)
            System.out.println(n + " is an Armstrong Number");
        else
            System.out.println(n + " is not an Armstrong Number");
    }
}
```

- With these corrections, the program will correctly check if a number is an Armstrong number.
- For an input of 153, the output will be : 153 is an Armstrong Number

Debug with correction:



Program-2: GCD_LCM.java

Code Debugging Analysis:

1. **Errors in the Code:**
   - **GCD Logic Error:** The condition `while(a % b == 0)` was not correctly handling the remainder check in the Euclidean algorithm. This needed to be changed to `while(a % b != 0)` to ensure that the loop runs until b divides a exactly.
   - **LCM Condition Error:** The LCM condition had to be modified. The correct check for LCM is `if(a % x == 0 && a % y == 0)` to ensure a is a multiple of both x and y.
2. **Breakpoints Required:**
   - **Breakpoint 1:** Set at the `while` loop in the GCD method to track whether the computation proceeds correctly based on the updated condition.
   - **Breakpoint 2:** Set at the `if` statement inside the LCM method to ensure that it correctly identifies the LCM.
   - **Breakpoint 3:** Set at the point where a is incremented in the LCM method to catch any off-by-one issues and prevent skipping over the correct LCM value.

   (A) Steps Taken to Fix the Errors:
   - Updated the logic in the GCD method by fixing the condition for the remainder, ensuring it matches the Euclidean algorithm's requirements.

- Corrected the LCM method by adjusting the condition to properly check for the least common multiple of the given values.

Debug without correction:

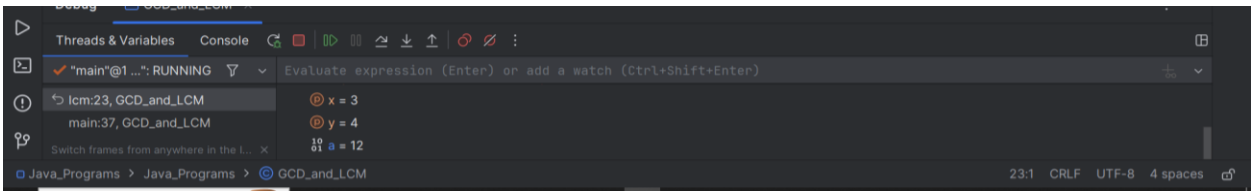
3. **Complete Executable Code:**

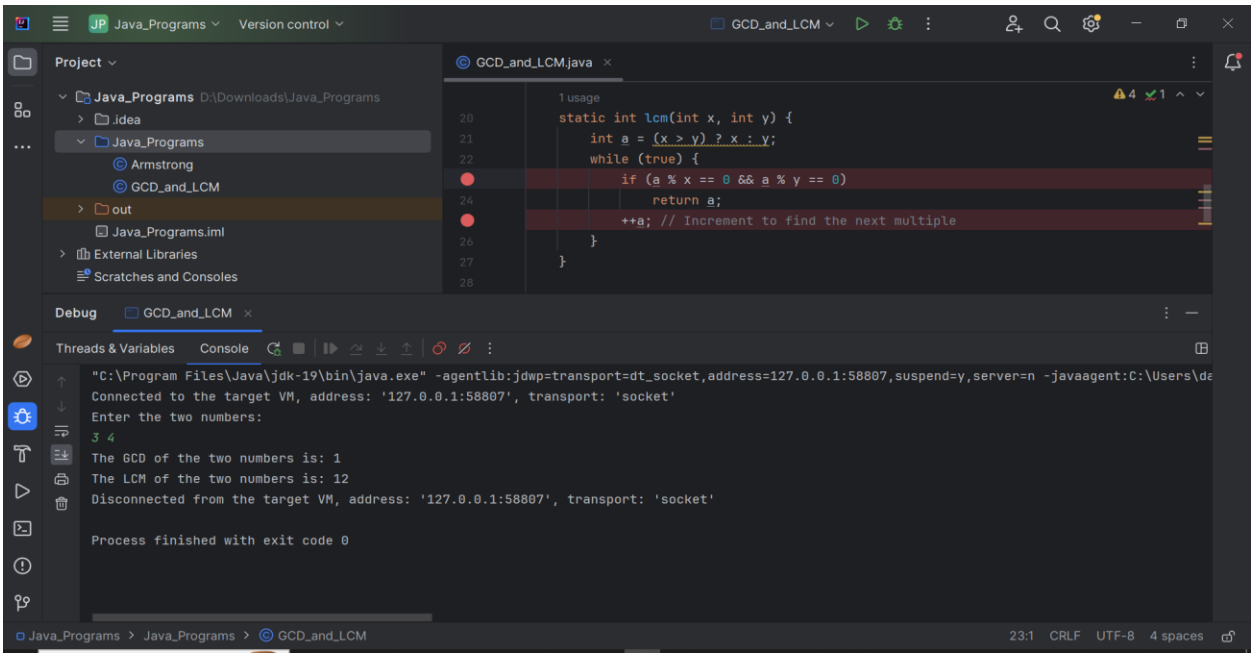
```
import java.util.Scanner;

public class GCD_LCM {

    // Function to calculate GCD using the Euclidean algorithm
    static int gcd(int x, int y) {
        int r, a, b;
        a = (x > y) ? x : y; // a is greater number
        b = (x < y) ? x : y; // b is smaller number

        while (b != 0) {
            r = a % b;
            a = b;
            b = r;
        }
        return a; // Return the GCD
    }

    // Function to calculate LCM
    static int lcm(int x, int y) {
        int a = (x > y) ? x : y;
        while (true) {
            if (a % x == 0 && a % y == 0)
                return a;
            ++a; // Increment to find the next multiple
        }
    }

    public static void main(String args[]) {
```

```
        Scanner input = new Scanner(System.in);
        System.out.println("Enter the two numbers: ");
        int x = input.nextInt();
        int y = input.nextInt();

        // Display GCD and LCM
        System.out.println("The GCD of the two numbers is: " + gcd(x, y));
        System.out.println("The LCM of the two numbers is: " + lcm(x, y));
        input.close();
    }
}
```

With these corrections, the program will correctly compute the GCD and LCM. The errors were mainly **computation errors (Category C)**.

Debug with corrected code:





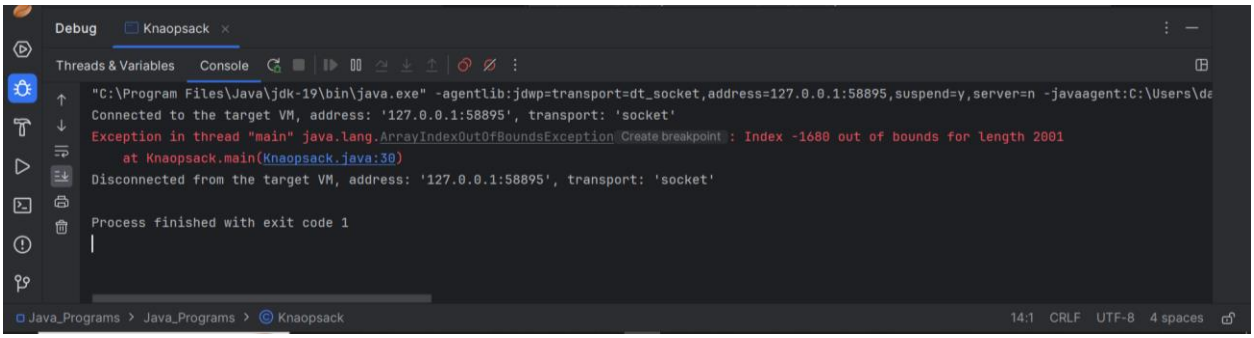
Program-3: Knapsack.java

Code Debugging Analysis:

1. **Errors in the Code:**
    - **Index Handling Issue:** The incorrect use of `n++` and `profit[n-2]` led to issues in array access and the dynamic programming logic. These needed to be corrected to maintain accurate indexing throughout the computation.
    - **Weight Condition Error:** The weight comparison logic `if (weight[n] > w)` was faulty and needed modification to ensure that the item is only considered if it fits within the current weight capacity.
2. **Breakpoints Required:**
    - **Breakpoint 1:** Set at `option1 = opt[n][w];` to verify that the correct index is being used when calculating the first option.
    - **Breakpoint 2:** Set at `option2 = profit[n] + opt[n-1][w-weight[n]];` to confirm that the second option is being computed correctly, considering the profit and reduced weight.
    - **Breakpoint 3:** Set at `if (weight[n] <= w)` to ensure the weight condition is properly evaluated before deciding whether to include the item.

    (A) Steps Taken to Fix the Errors:
    - Resolved the index handling issue by replacing `n++` with `n` in `option1 = opt[n][w];` to maintain proper array access.
    - Corrected the profit calculation by changing `profit[n-2]` to `profit[n]` for accurate profit reference.
    - Fixed the weight comparison logic by updating it to `if (weight[n] <= w)` to ensure the item is only taken if its weight does not exceed the current capacity.

Debug without correction:

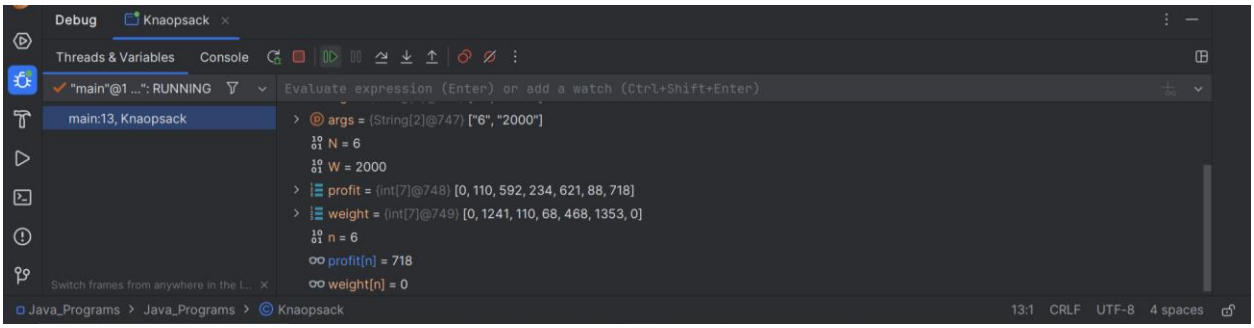


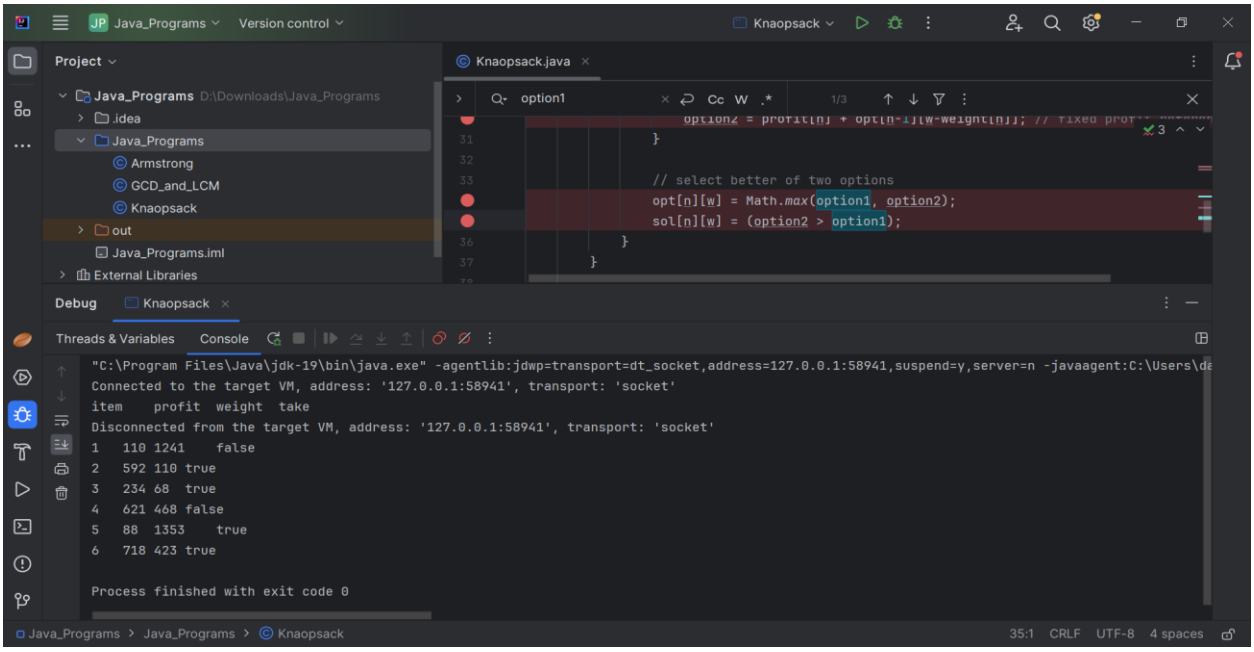
3. Complete Executable Code:

```
public class Knapsack {

   public static void main(String[] args) {
      int N = Integer.parseInt(args[0]);   // number of items
      int W = Integer.parseInt(args[1]);   // maximum weight of knapsack

      int[] profit = new int[N+1];
      int[] weight = new int[N+1];

      // generate random instance, items 1..N
      for (int n = 1; n <= N; n++) {
         profit[n] = (int) (Math.random() * 1000);
         weight[n] = (int) (Math.random() * W);
      }

      // opt[n][w] = max profit of packing items 1..n with weight limit w
      // sol[n][w] = does the solution to pack items 1..n with weight limit w include item n?
      int[][] opt = new int[N+1][W+1];
      boolean[][] sol = new boolean[N+1][W+1];

      for (int n = 1; n <= N; n++) {
         for (int w = 1; w <= W; w++) {

            // don't take item n
            int option1 = opt[n][w];  // fixed from n++

            // take item n
            int option2 = Integer.MIN_VALUE;
            if (weight[n] <= w) {  // fixed weight check
               option2 = profit[n] + opt[n-1][w-weight[n]]; // fixed profit reference
            }

            // select better of two options
            opt[n][w] = Math.max(option1, option2);
            sol[n][w] = (option2 > option1);
         }
      }

      // determine which items to take
      boolean[] take = new boolean[N+1];
      for (int n = N, w = W; n > 0; n--) {
         if (sol[n][w]) {
            take[n] = true;
```

```
            w = w - weight[n];
        } else {
            take[n] = false;
        }
    }

    // print results
    System.out.println("item" + "\t" + "profit" + "\t" + "weight" + "\t" + "take");
    for (int n = 1; n <= N; n++) {
        System.out.println(n + "\t" + profit[n] + "\t" + weight[n] + "\t" + take[n]);
    }
  }
}
```

Debug after correction:





Program-4: MagicNumberCheck.java

Code Debugging Analysis:

1. **Errors in the Code:**
    - **Inner Loop Condition:** The inner loop isn't iterating over each digit of the sum correctly due to an incorrect condition. This is preventing the loop from functioning as intended.
    - **Incorrect Product Instead of Sum:** Instead of adding the digits, the code was incorrectly multiplying them, which is a logic error.
    - **Missing Semicolon:** There was a syntax error where a semicolon was missing, which needs to be corrected.
2. **Breakpoints Required:**
    - **Breakpoint 1:** Set at the start of the inner `while` loop to confirm whether the sum is processed properly through digit extraction.
    - **Breakpoint 2:** Set at the line `s = s + (sum % 10);` to ensure that each digit is being added correctly to the sum.
    - **Breakpoint 3:** Set at the condition `if(num == 1)` to check if the final number is evaluated as expected.
3. **Steps Taken to Fix the Errors:**
    - Modified the inner loop condition from `while(sum == 0)` to `while(sum > 0)` to ensure that the loop iterates as long as there are digits left to process.
    - Corrected the logic by changing `s = s * (sum / 10);` to `s = s + (sum % 10);` to ensure the sum of the digits is calculated.
    - Added the missing semicolon after `sum = sum % 10;` to resolve the syntax error.

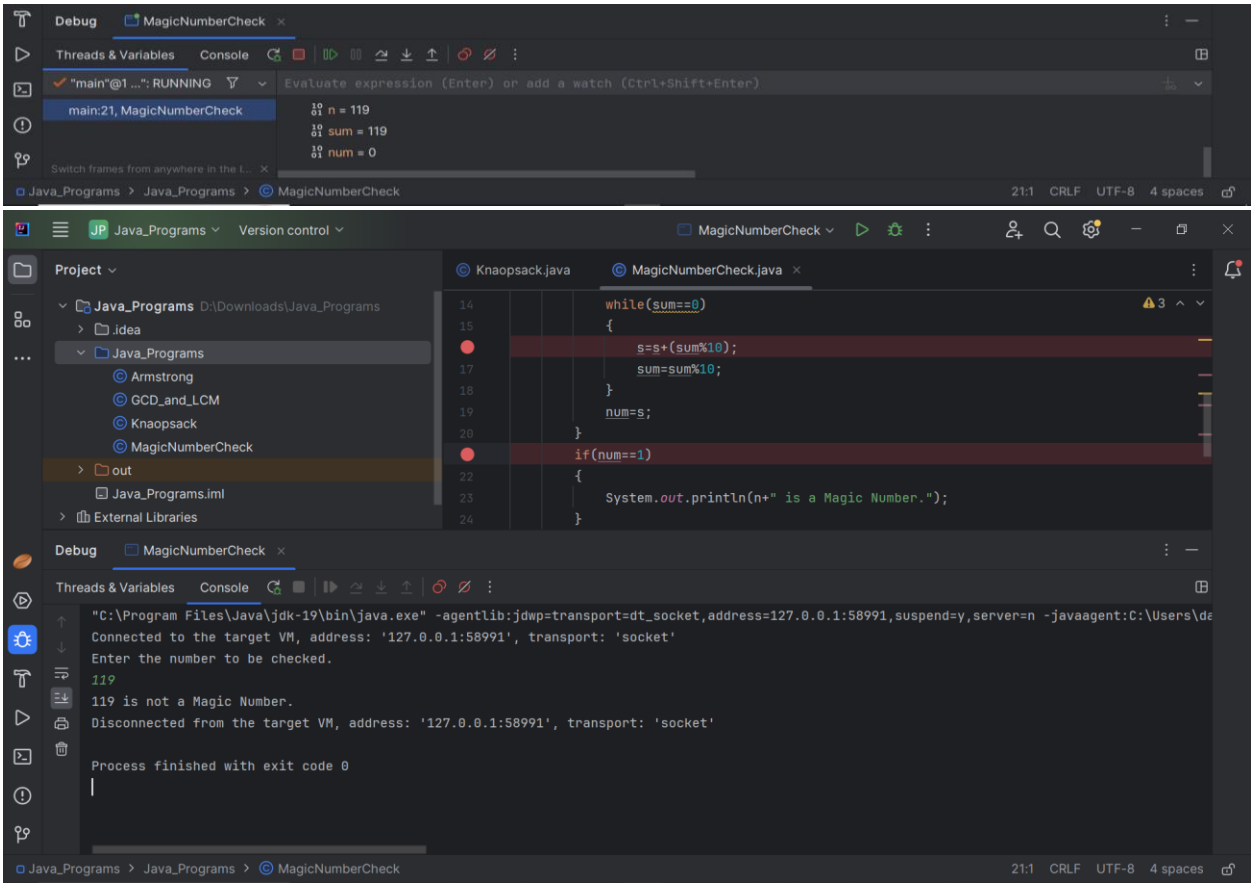
Debug before correction:

3. Complete Executable Code:

```
import java.util.*;
public class MagicNumberCheck {
    public static void main(String args[]) {
        Scanner ob = new Scanner(System.in);
```

```
        System.out.println("Enter the number to be checked.");
        int n = ob.nextInt();
        int sum = 0, num = n;
        while (num > 9) {
            sum = num;
            int s = 0;
            while (sum > 0) {  // fixed the while condition
                s = s + (sum % 10);  // fixed the sum calculation
                sum = sum / 10;  // fixed digit extraction
            }
            num = s;
        }
        if (num == 1) {
            System.out.println(n + " is a Magic Number.");
        } else {
            System.out.println(n + " is not a Magic Number.");
        }
        ob.close();
    }
}
```

Debug after correction:

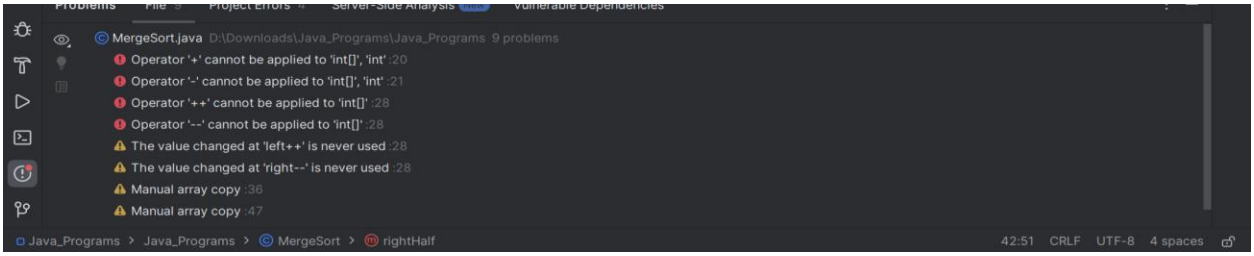


Program-5: MergeSort.java

Code Debugging Analysis:

1. **Errors in the Code:**
   - **Array Slicing for Left and Right Halves:** The logic for slicing the array does not correctly divide it into the left and right halves, leading to incorrect array splitting.
   - **Incorrect Merge Call:** The merging syntax is incorrect due to improper use of increment and decrement operators, which disrupts the merging process.
2. **Breakpoints Required:**
   - **Breakpoint 1:** Place at the start of the `mergeSort` method to verify that the array is properly divided into left and right halves.
   - **Breakpoint 2:** Place at the `merge` method to ensure that the merging process is working correctly when combining the sorted halves.
   - **Breakpoint 3:** Place at the points where the `leftHalf` and `rightHalf` arrays are created to check if the array slicing is functioning as expected.
3. **Steps Taken to Fix the Errors:**
   - Updated the slicing logic in the `mergeSort` method to use `Arrays.copyOfRange` for correctly splitting the array into two halves.
   - Removed the `++` and `--` operators from the merge call and used a simpler `merge(array, left, right);` to prevent the misuse of increment/decrement logic.

Debug before correction:

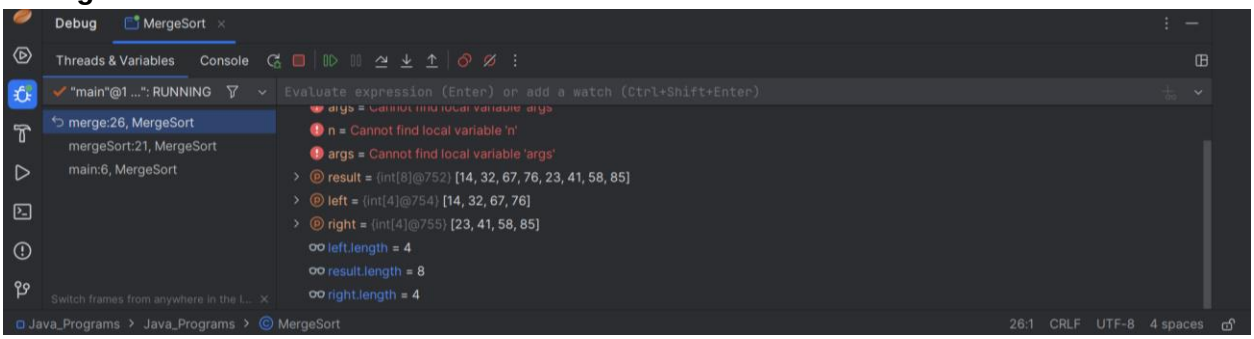


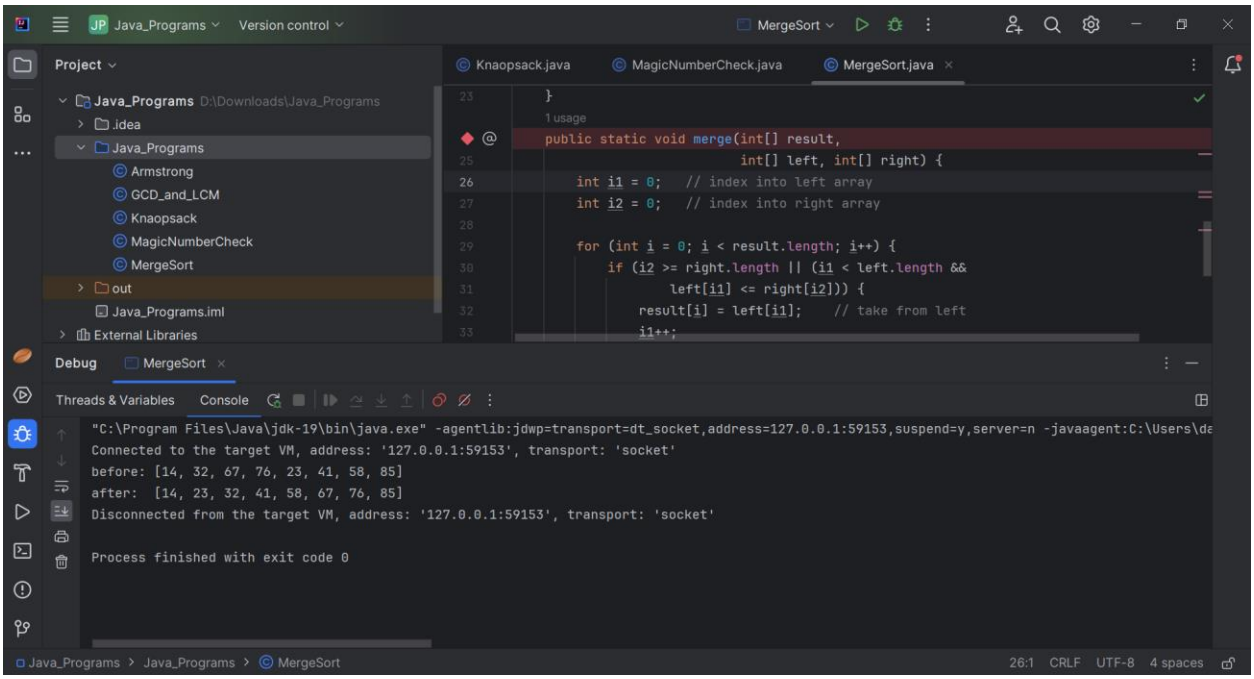
3. **Complete Executable Code:**

```
import java.util.*;
public class MergeSort {
   public static void main(String[] args) {
      int[] list = {14, 32, 67, 76, 23, 41, 58, 85};
      System.out.println("before: " + Arrays.toString(list));
      mergeSort(list);
      System.out.println("after:  " + Arrays.toString(list));
   }
   public static void mergeSort(int[] array) {
      if (array.length > 1) {
```

```
        // split array into two halves
        int mid = array.length / 2;
        int[] left = Arrays.copyOfRange(array, 0, mid); // corrected slicing
        int[] right = Arrays.copyOfRange(array, mid, array.length); // corrected slicing

        // recursively sort the two halves
        mergeSort(left);
        mergeSort(right);

        // merge the sorted halves into a sorted whole
        merge(array, left, right); // fixed merge call
    }
  }
  public static void merge(int[] result,
                           int[] left, int[] right) {
    int i1 = 0;   // index into left array
    int i2 = 0;   // index into right array

    for (int i = 0; i < result.length; i++) {
      if (i2 >= right.length || (i1 < left.length &&
          left[i1] <= right[i2])) {
        result[i] = left[i1];    // take from left
        i1++;
      } else {
        result[i] = right[i2];   // take from right
        i2++;
      }
    }
  }
}
```

Debug after correction:





Program-6: MatrixMultiplication.java

Code Debugging Analysis:

1. **Errors in the Code:**
   - **Indexing Error in Matrix Multiplication Logic:** The indices used in the matrix multiplication process were incorrect, leading to potential errors in accessing matrix elements.
   - **Incorrect Input Prompt:** The input prompt for the dimensions of the second matrix was incorrectly labeled as pertaining to the first matrix, which could confuse the user.
2. **Breakpoints Required:**
   - **Breakpoint 1:** Set within the multiplication loop to observe the index values and ensure that the correct elements of the matrices are being accessed during the multiplication.
   - **Breakpoint 2:** Set right before printing the resulting matrix to verify that the final output contains the expected values.
3. **Steps Taken to Fix the Errors:**
   - Adjusted the multiplication logic to use the correct indices: `first[c][k]` and `second[k][d]` to ensure accurate matrix calculations.
   - Updated the second input prompt to clarify that the user should input the dimensions for the second matrix, eliminating any potential confusion.

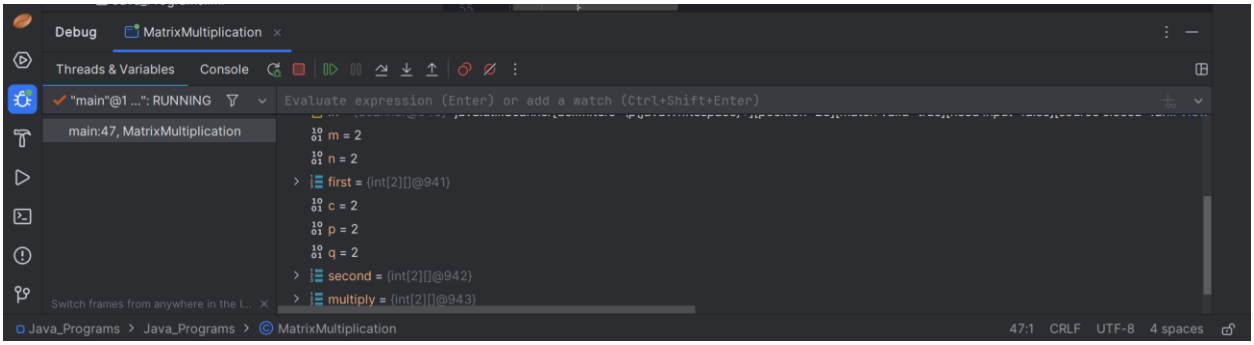
Debug before correction:

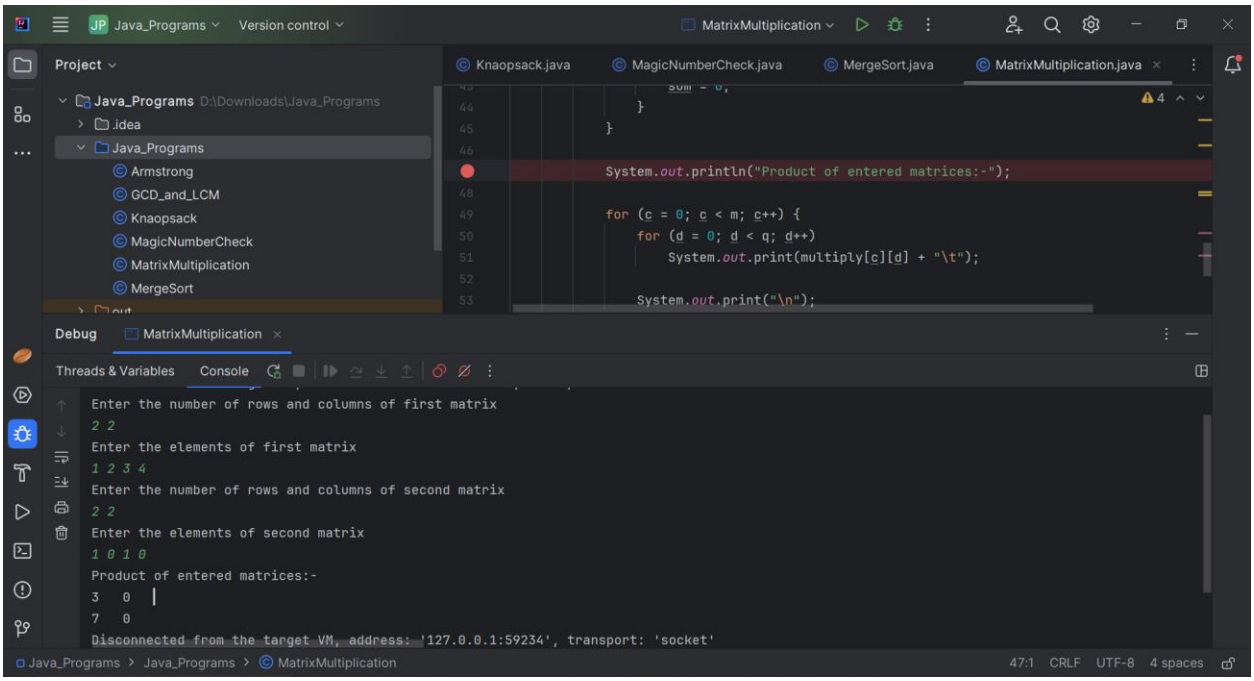
3. Complete Executable Code:

```
import java.util.Scanner;

class MatrixMultiplication {
   public static void main(String args[]) {
      int m, n, p, q, sum = 0, c, d, k;

      Scanner in = new Scanner(System.in);
      System.out.println("Enter the number of rows and columns of first matrix");
      m = in.nextInt();
      n = in.nextInt();

      int first[][] = new int[m][n];

      System.out.println("Enter the elements of first matrix");

      for (c = 0; c < m; c++)
         for (d = 0; d < n; d++)
            first[c][d] = in.nextInt();

      System.out.println("Enter the number of rows and columns of second matrix"); // Fixed
prompt
      p = in.nextInt();
      q = in.nextInt();

      if (n != p)
         System.out.println("Matrices with entered orders can't be multiplied with each other.");
      else {
         int second[][] = new int[p][q];
         int multiply[][] = new int[m][q];

         System.out.println("Enter the elements of second matrix");
```

```
        for (c = 0; c < p; c++)
            for (d = 0; d < q; d++)
                second[c][d] = in.nextInt();

        for (c = 0; c < m; c++) {
            for (d = 0; d < q; d++) {
                for (k = 0; k < n; k++) { // Corrected loop for proper multiplication
                    sum = sum + first[c][k] * second[k][d]; // Corrected indexing
                }

                multiply[c][d] = sum;
                sum = 0;
            }
        }

        System.out.println("Product of entered matrices:-");

        for (c = 0; c < m; c++) {
            for (d = 0; d < q; d++)
                System.out.print(multiply[c][d] + "\t");

            System.out.print("\n");
        }
    }
    in.close();
  }
}
```

Debug after correction:





Program-7: QuadraticProbingHashTable.java

Code Debugging Analysis:

1. **Errors in the Code:**
   - **Syntax Error in Insert Method:** The addition assignment operator was improperly formatted, causing a syntax error.
   - **Incorrect Probing Logic:** The calculation of the probe index did not adhere to the rules of quadratic probing, which could lead to incorrect index generation.
   - **Potential Array Index Out of Bounds:** The way array indices are manipulated could lead to accessing invalid positions in the array, potentially causing runtime errors.
2. **Breakpoints Required:**
   - **Breakpoint 1:** Set within the insert method to track the values of `i` and `h` during insertion operations, allowing for closer examination of how the indices are being computed.
   - **Breakpoint 2:** Set in the get method to monitor the hash values that are being accessed, ensuring that the correct values are retrieved.

   (A) Steps Taken to Fix the Errors:
   - Fixed the syntax error in the insert method by correcting the formatting of the addition assignment operator.
   - Adjusted the probing logic to properly implement the quadratic probing technique, ensuring accurate index calculations.
   - Implemented checks to prevent out-of-bounds access when manipulating array indices, thereby enhancing the robustness of the code.

Debug before correction:

3. Complete Executable Code:

```
import java.util.Scanner;

/** Class QuadraticProbingHashTable **/
class QuadraticProbingHashTable {
    private int currentSize, maxSize;
    private String[] keys;
    private String[] vals;

    /** Constructor **/
    public QuadraticProbingHashTable(int capacity) {
        currentSize = 0;
        maxSize = capacity;
        keys = new String[maxSize];
        vals = new String[maxSize];
    }

    /** Function to clear hash table **/
    public void makeEmpty() {
        currentSize = 0;
        keys = new String[maxSize];
        vals = new String[maxSize];
    }

    /** Function to get size of hash table **/
    public int getSize() {
        return currentSize;
    }

    /** Function to check if hash table is full **/
    public boolean isFull() {
        return currentSize == maxSize;
    }
```

```
/** Function to check if hash table is empty **/
public boolean isEmpty() {
    return getSize() == 0;
}

/** Function to check if hash table contains a key **/
public boolean contains(String key) {
    return get(key) != null;
}

/** Function to get hash code of a given key **/
private int hash(String key) {
    return (key.hashCode() % maxSize + maxSize) % maxSize; // Ensuring non-negative index
}

/** Function to insert key-value pair **/
public void insert(String key, String val) {
    if (isFull()) {
        System.out.println("Hash Table is full. Cannot insert.");
        return;
    }

    int tmp = hash(key);
    int i = tmp, h = 1;

    do {
        if (keys[i] == null) {
            keys[i] = key;
            vals[i] = val;
            currentSize++;
            return;
        }
        if (keys[i].equals(key)) {
            vals[i] = val;
            return;
        }
        i = (tmp + h * h) % maxSize; // Corrected probing logic
        h++;
    } while (i != tmp);
}

/** Function to get value for a given key **/
public String get(String key) {
    int i = hash(key), h = 1;
```

```
        while (keys[i] != null) {
            if (keys[i].equals(key))
                return vals[i];
            i = (hash(key) + h * h) % maxSize; // Corrected probing logic
            h++;
        }
        return null;
    }

    /** Function to remove key and its value **/
    public void remove(String key) {
        if (!contains(key))
            return;

        /** find position key and delete **/
        int i = hash(key), h = 1;
        while (!key.equals(keys[i])) {
            i = (hash(key) + h * h) % maxSize;
            h++;
        }

        keys[i] = vals[i] = null;

        /** rehash all keys **/
        for (i = (i + h * h) % maxSize; keys[i] != null; i = (i + h * h) % maxSize) {
            String tmp1 = keys[i], tmp2 = vals[i];
            keys[i] = vals[i] = null;
            currentSize--;
            insert(tmp1, tmp2);
        }
        currentSize--;
    }

    /** Function to print HashTable **/
    public void printHashTable() {
        System.out.println("\nHash Table: ");
        for (int i = 0; i < maxSize; i++)
            if (keys[i] != null)
                System.out.println(keys[i] +" "+ vals[i]);
        System.out.println();
    }
}
```

```
/** Class QuadraticProbingHashTableTest **/
public class QuadraticProbingHashTableTest {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        System.out.println("Hash Table Test\n\n");
        System.out.println("Enter size");
        /** make object of QuadraticProbingHashTable **/
        QuadraticProbingHashTable qpht = new QuadraticProbingHashTable(scan.nextInt());

        char ch;
        /**  Perform QuadraticProbingHashTable operations  **/
        do {
            System.out.println("\nHash Table Operations\n");
            System.out.println("1. insert ");
            System.out.println("2. remove");
            System.out.println("3. get");
            System.out.println("4. clear");
            System.out.println("5. size");

            int choice = scan.nextInt();
            switch (choice) {
                case 1 :
                    System.out.println("Enter key and value");
                    qpht.insert(scan.next(), scan.next());
                    break;
                case 2 :
                    System.out.println("Enter key");
                    qpht.remove(scan.next());
                    break;
                case 3 :
                    System.out.println("Enter key");
                    System.out.println("Value = "+ qpht.get(scan.next()));
                    break;
                case 4 :
                    qpht.makeEmpty();
                    System.out.println("Hash Table Cleared\n");
                    break;
                case 5 :
                    System.out.println("Size = "+ qpht.getSize());
                    break;
                default :
                    System.out.println("Wrong Entry \n ");
                    break;
            }
```

```
            /** Display hash table **/
            qpht.printHashTable();

            System.out.println("\nDo you want to continue (Type y or n) \n");
            ch = scan.next().charAt(0);
        } while (ch == 'Y' || ch == 'y');
        scan.close();
    }
}
```

Program-8: Ascending_Order.java

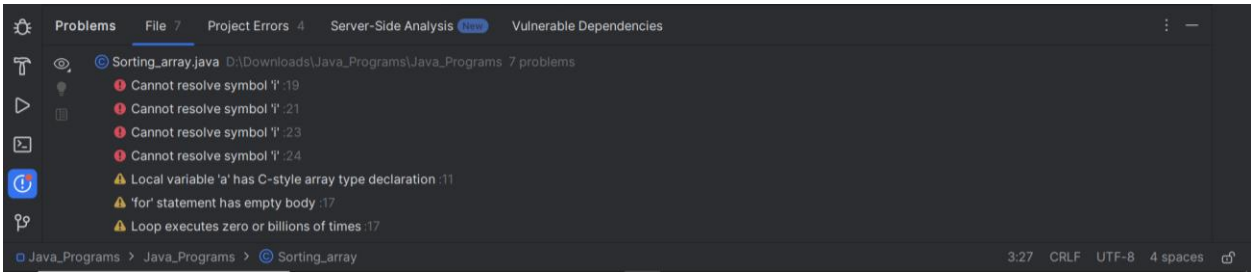
Code Debugging Analysis:

1. **Errors in the Code:**
    - The loop condition and the sorting logic were flawed, resulting in incorrect sorting behavior.
2. **Breakpoints Required:**
    - **Breakpoint 1:** Set a breakpoint at the loop header `for (int i = 0; i < n; i++)` to confirm that the loop iterates as expected.
    - **Breakpoint 2:** Set a breakpoint within the inner loop to verify that the sorting condition is being evaluated correctly.

(A) Steps Taken to Fix the Errors:

- Removed the semicolon that was incorrectly placed at the end of the first `for` loop, which was causing the loop to terminate prematurely.
- Updated the condition in the sorting logic to `if (a[i] > a[j])` to ensure that the sorting is performed correctly by comparing the elements in the appropriate manner.

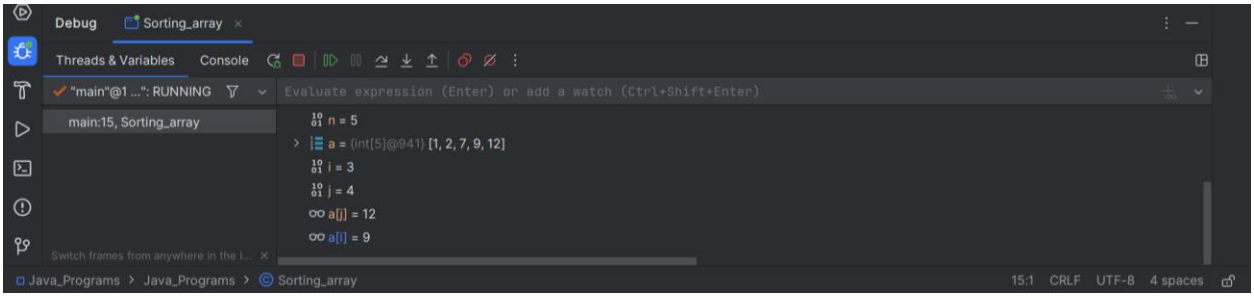
Debug before correction:

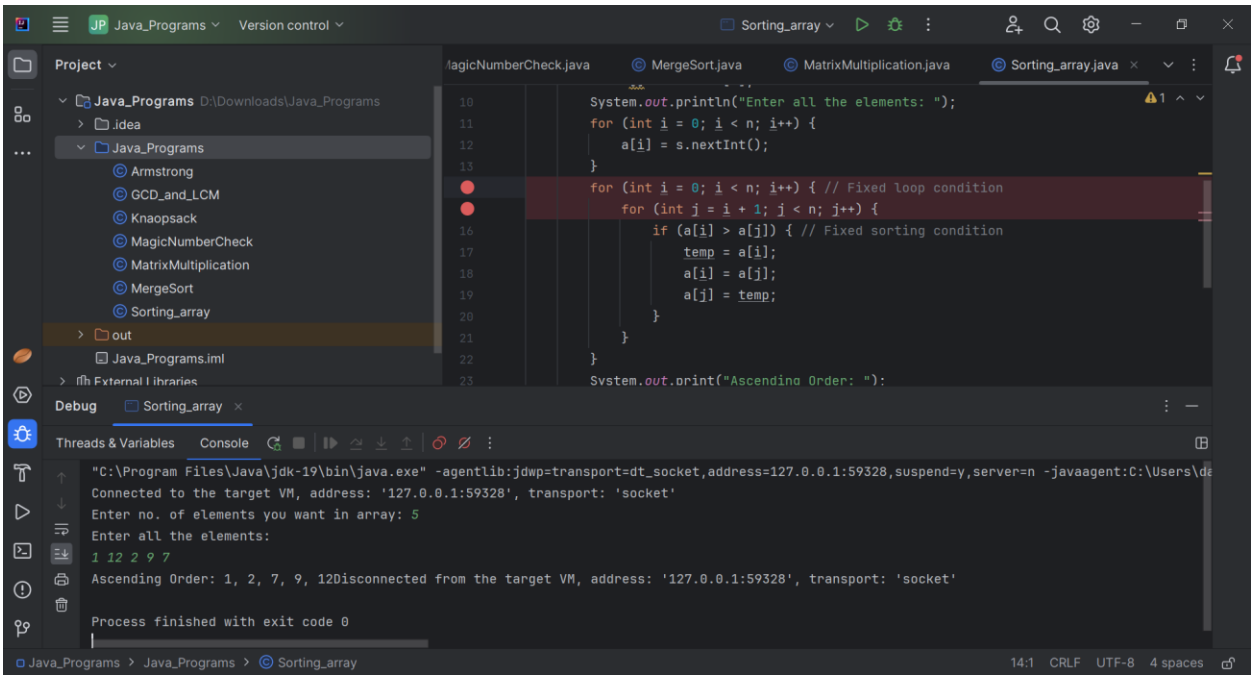


3. **Complete Executable Code:**

```
import java.util.Scanner;

public class Ascending_Order {
    public static void main(String[] args) {
        int n, temp;
        Scanner s = new Scanner(System.in);
        System.out.print("Enter no. of elements you want in array: ");
        n = s.nextInt();
        int a[] = new int[n];
        System.out.println("Enter all the elements: ");
        for (int i = 0; i < n; i++) {
            a[i] = s.nextInt();
        }
        for (int i = 0; i < n; i++) { // Fixed loop condition
            for (int j = i + 1; j < n; j++) {
                if (a[i] > a[j]) { // Fixed sorting condition
                    temp = a[i];
                    a[i] = a[j];
                    a[j] = temp;
                }
            }
        }
        System.out.print("Ascending Order: ");
        for (int i = 0; i < n - 1; i++) {
            System.out.print(a[i] + ", ");
        }
        System.out.print(a[n - 1]);
    }
}
```

Debug after correction:





Program-9: StackMethods.java

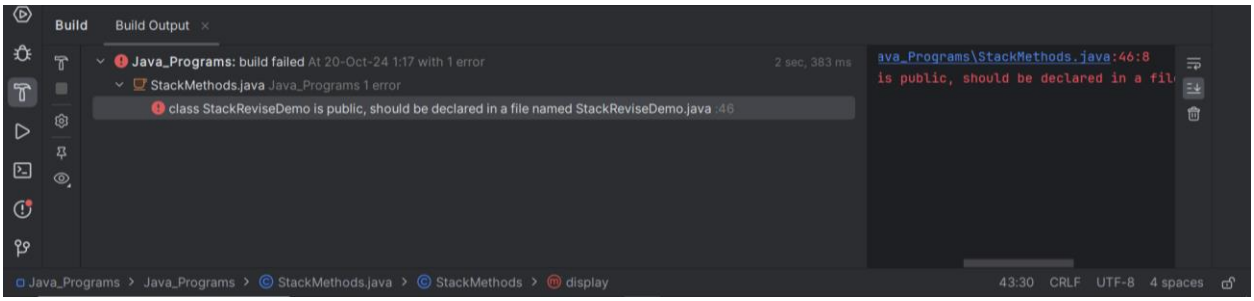
Code Debugging Analysis:

1. **Errors in the Code:**
    - The `push`, `pop`, and `display` methods contained incorrect indexing logic, leading to improper stack operations.
2. **Breakpoints Required:**
    - **Breakpoint 1:** Set a breakpoint in the `push` method to verify that values are being added to the stack correctly.
    - **Breakpoint 2:** Set a breakpoint in the `pop` method to confirm that values are being removed from the stack as intended.
    - **Breakpoint 3:** Set a breakpoint in the `display` method to examine the output of the stack's contents.

(A) Steps Taken to Fix the Errors:

- Changed `top--;` to `top++;` in the `push` method to correctly increment the index when a value is added.
- Changed `top++;` to `top--;` in the `pop` method to correctly decrement the index when a value is removed.
- Updated the loop condition in the `display` method to `i <= top` to ensure all elements in the stack are displayed properly.

Debug before correction:



3. **Complete Executable Code:**

```
// Stack implementation in Java
import java.util.Arrays;

public class StackMethods {
    private int top;
    int size;
    int[] stack;

    public StackMethods(int arraySize) {
        size = arraySize;
        stack = new int[size];
        top = -1;
    }

    public void push(int value) {
        if (top == size - 1) {
            System.out.println("Stack is full, can't push a value");
        } else {
            top++; // Increment top to add value
            stack[top] = value; // Assign value to stack
        }
    }

    public void pop() {
        if (!isEmpty()) {
            System.out.println("Popped value: " + stack[top]); // Optional: Print popped value
            top--; // Decrement top to remove value
        } else {
            System.out.println("Can't pop...stack is empty");
        }
    }
```

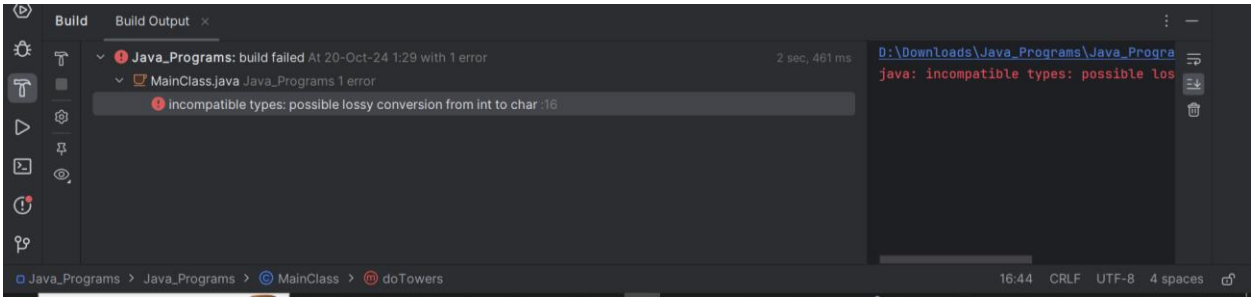
```
    public boolean isEmpty() {
        return top == -1;
    }
    public void display() {
        if (isEmpty()) {
            System.out.println("Stack is empty");
            return;
        }
        for (int i = 0; i <= top; i++) { // Changed condition to i <= top
            System.out.print(stack[i] + " ");
        }
        System.out.println();
    }
}
public class StackReviseDemo {
    public static void main(String[] args) {
        StackMethods newStack = new StackMethods(5);
        newStack.push(10);
        newStack.push(1);
        newStack.push(50);
        newStack.push(20);
        newStack.push(90);

        newStack.display(); // Should print all stack elements
        newStack.pop(); // Popping elements
        newStack.pop();
        newStack.pop();
        newStack.pop();
        newStack.display(); // Display remaining elements in the stack
    }
}
```

Program-10: Tower of Hanoi.java

Code Debugging Analysis:

1. **Errors in the Code:**
    - The main recursive function contains issues with parameter handling during the recursive calls, which could lead to incorrect behavior.
2. **Breakpoints Required:**
    - **Breakpoint 1:** Set a breakpoint at the start of the `doTowers` method to examine the parameter values during each recursive call.
    - **Breakpoint 2:** Set a breakpoint before each print statement to verify the current state of the disks and the moves being executed.
3. **Steps Taken to Fix the Errors:**
    - Modified the recursive calls to ensure that parameters are passed correctly without the use of increment or decrement operators, thus preserving their intended values during recursion.

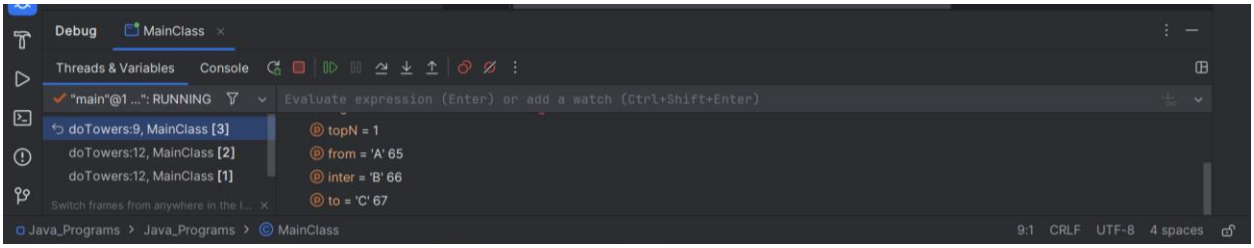
Debug before correction:

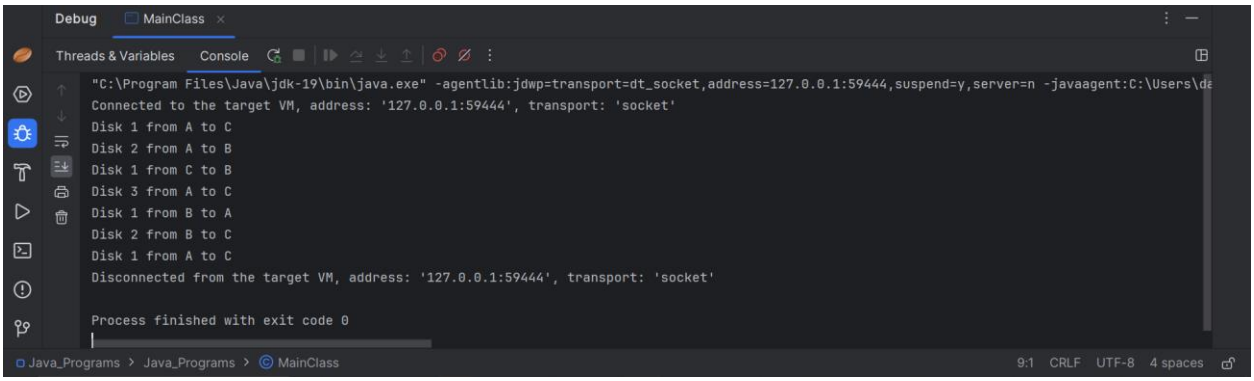


3. **Complete Executable Code:**

```
// Tower of Hanoi
public class MainClass {
  public static void main(String[] args) {
    int nDisks = 3; // Number of disks
    doTowers(nDisks, 'A', 'B', 'C'); // A, B, and C are the names of the rods
  }
  public static void doTowers(int topN, char from, char inter, char to) {
    if (topN == 1) {
      System.out.println("Disk 1 from " + from + " to " + to);
    } else {
      doTowers(topN - 1, from, to, inter); // Move topN-1 disks from source to intermediate
      System.out.println("Disk " + topN + " from " + from + " to " + to); // Move the last disk
      doTowers(topN - 1, inter, from, to); // Move the topN-1 disks from intermediate to target
    }
  }
```

Debug after correction:





[Q-3] Static Analysis Tools:

Repository Link : https://github.com/HouariZegai/Calculator

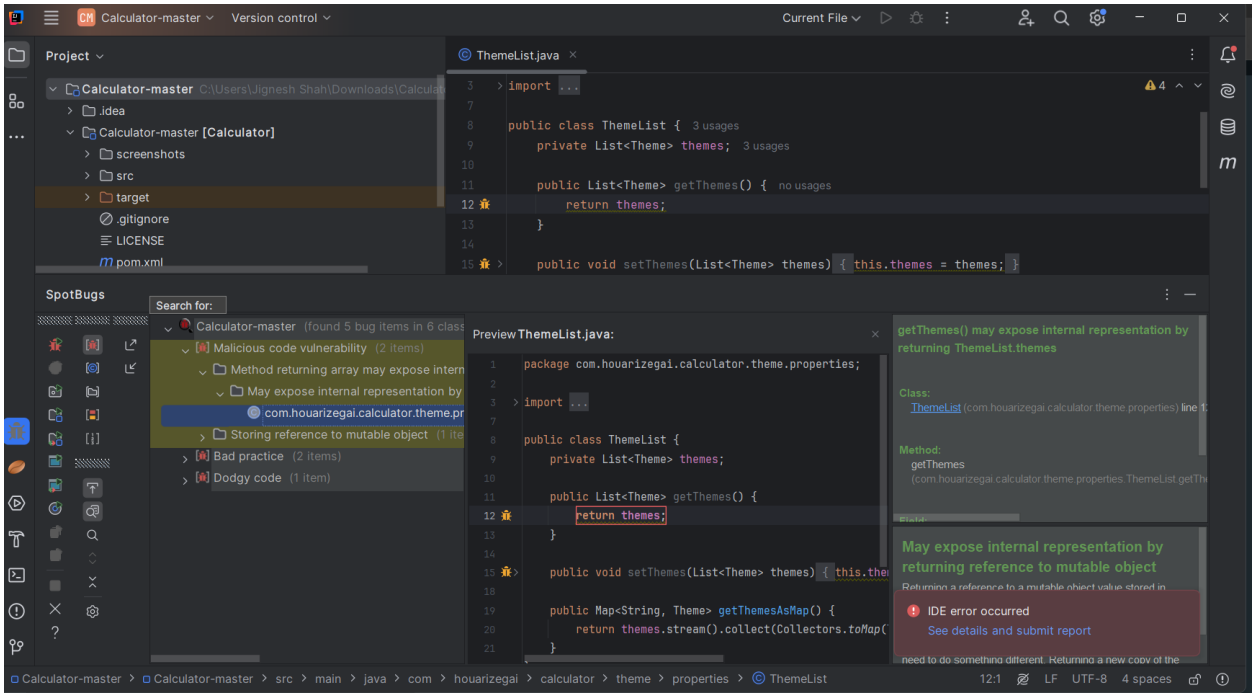
Tool Used:

SpotBugs:

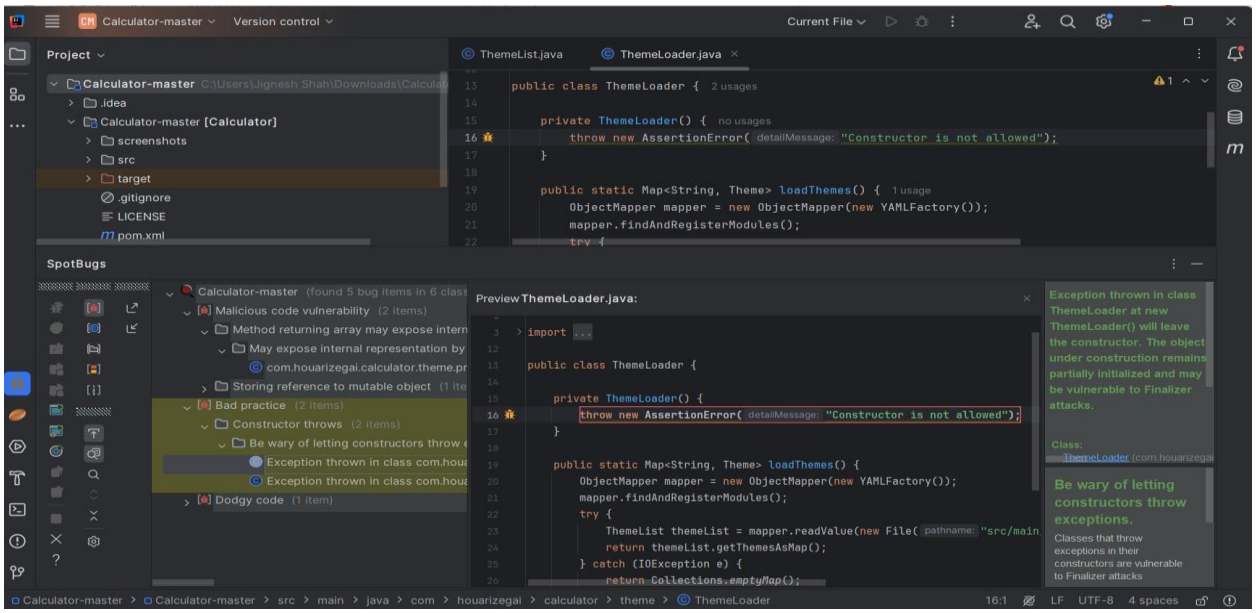
- SpotBugs is a powerful static analysis tool that helps Java developers identify potential bugs and quality issues in their code. With its customizable checks, detailed reports, and integration capabilities, it is a valuable addition to any Java development workflow, aiding in the creation of more reliable and maintainable software.

ISSUES IDENTIFY:

(1) May expose the internal represenation by returning the reference to mutable object



(2) Bad Practice



(3) Dodgy code