Finalexampractice Bugsummary

1 Bug Summary - Quick Reference

1.1 Critical Bugs Found and Fixed

1.1.1 1. **Line 60: Missing Stack Initialization**

```
// BUG:
ldi d,

// FIX:
ldi d, 0        // Initialize stack pointer to 0
```

1.1.2 2. **Line 107: Wrong Type Size for fPtr**

```
// BUG:
isLoop_result: isLoop_fPtr Node_size +

// FIX:
isLoop_result: isLoop_fPtr 1 +  // fPtr is a pointer (1 byte), not a Node (2 bytes)
```

1.1.3 3. **Line 112: Invalid Syntax for Allocation**

```
// BUG:
ldi c,0 isLoop_lvs -
add d,c

// FIX:
ldi c, isLoop_lvs  // Load local var size
sub d, c            // Allocate locals (subtract from stack pointer)
```

1.1.4 4. **Lines 119-121: Incorrect Parameter Access**

```
// BUG:
ldi a,isLoop_pNode
ld a,(a)           // Tries to dereference offset, not parameter!

// FIX:
ldi a, isLoop_pNode // A = offset to parameter
add a, d            // A = address of parameter
ld a, (a)           // A = value of parameter
```

1.1.5 5. **Lines 124-129: Incorrect prevFrame Assignment**

```
// BUG:
ldi a,isLoop_prevFrame
add a,d
ldi c,Frame_prevFrame Frame_pNode -  // Wrong calculation
add b,c

// FIX:
ldi a, isLoop_prevFrame // A = offset to prevFrame parameter
add a, d                // A = address of prevFrame parameter
ld a, (a)               // A = prevFrame value
ldi b, isLoop_f
add b, Frame_prevFrame  // B = offset to f.prevFrame
add b, d                // B = address of f.prevFrame
st (b), a               // f.prevFrame = prevFrame
```

1.1.6 6. **Lines 136-137: Wrong Parameter Access (Struct Member vs Parameter)**

```
// BUG:
ldi a,Frame_prevFrame  // This is struct MEMBER offset, not parameter!
ld a,(a)               // Wrong!

// FIX:
ldi a, isLoop_prevFrame // A = offset to prevFrame PARAMETER
add a, d                // A = address of prevFrame parameter
ld a, (a)               // A = prevFrame parameter value
```

1.1.7 7. **Lines 156-162: Wrong Offset for pNode Parameter**

```
// BUG:
ldi c,Frame_pNode 1 +  // This gives 1, not parameter offset!
add c,d                // C = D+1 (points to f.prevFrame, not pNode!)
ld c,(c)               // Loads wrong value

// FIX:
ldi c, isLoop_pNode // C = 5 (offset to pNode parameter)
inc c                // C = 6 (adjust for the push: dec d, st (d),c)
add c, d             // C = D+6 (address of pNode parameter)
ld c, (c)            // C = pNode parameter value
```

1.1.8 8. **Line 162: Wrong Jump Condition**

```
// BUG:
jzi isLoop_then0_while0_begin  // Loops forever if fPtr->pNode == pNode!

// FIX:
jzi isLoop_then0_while0_end    // Exit loop when fPtr->pNode == pNode
```

```
// BUG:
st (c),b  // Register c doesn't contain fPtr address here!

// FIX:
// After computing fPtr->prevFrame in register C:
// Register B contains address of fPtr variable (from earlier)
st (b), c  // Store new fPtr value back to fPtr variable
```

1.1.10 10. **Line 172: Register c Doesn't Contain fPtr**

```
// BUG:
and c,c  // Testing wrong register - c was used for other things!

// FIX:
ldi c, isLoop_fPtr  // C = offset to fPtr variable
add c, d            // C = address of fPtr variable
ld c, (c)           // C = fPtr value
and c, c            // Test if fPtr == 0
```

1.1.11 11. **Line 182: Wrong Offset for pNode in Recursive Call**

```
// BUG:
ldi b,isLoop_pNode 1 +  // 5+1=6, but after push it's wrong offset
add b,d
ld b,(b)

// FIX:
// After push: dec d, st (d),c (push &f)
// pNode was at D+5, now at D+6 (frame base moved from D to D+1)
ldi b, isLoop_pNode // B = 5
inc b               // B = 6 (adjust for push)
add b, d            // B = D+6 (address of pNode parameter)
ld b, (b)           // B = pNode parameter value
```

1.1.12 12. **Line 218: Missing Value for prevFrame Argument**

```
// BUG:
ldi a,  // No value!

// FIX:
ldi a, 0  // Push 0 as prevFrame argument
```

1.1.13 13. **Line 232: Missing Value for Argument Cleanup**

```
// BUG:
ldi b,  // No value!
add d,b

// FIX:
inc d  // Clean up first argument
inc d  // Clean up second argument
```

1.1.14 14. **Line 199: Wrong Label Name**

```
// BUG:
isLoop_endif0:  // But jump at line 132 goes to isLoop_endif

// FIX:
isLoop_endif:  // Match the label name used in jzi
```

1.2 Common Patterns to Watch For

1. **Parameter Access Pattern**:

```
    ldi reg, param_offset  // Load offset
    add reg, d             // Compute address (D is stack pointer)
    ld reg, (reg)          // Load value
```

1. **Local Variable Access Pattern**:

```
    ldi reg, local_offset  // Load offset (usually small, like 0, 1, 2...)
    add reg, d             // Compute address (D points to frame base)
    ld/st (reg), ...       // Access value
```

1. **Struct Member Access Pattern**:

```
    ldi reg, struct_base   // Load base address of struct
    add reg, member_offset // Add member offset (e.g., Frame_pNode = 0)
    ld/st (reg), ...       // Access member
```

1. **Stack Balance Check**:

- Before function call: count `dec d` (pushes) - After function call: count `inc d` (pops) - Should differ by exactly 1 (the return address, popped by callee)

1.3 Register Tracking Tips

- **Always track what each register contains** at each point
- **Write comments** showing the C code concept each register represents
- **Check for overwrites** before using a register value
- **Use temporary saves** on stack if needed during complex operations

---

1.4 Stack Frame Visualization

For `isLoop` function:

```
After allocation:
D+6: prevFrame (parameter 2)
D+5: pNode (parameter 1)
D+4: return address
```

```
D+2: fPtr (local variable)
D+1: f.prevFrame (member of local struct f)
D+0: f.pNode (member of local struct f) ← D points here
```

Key insight: **Offsets are always relative to where D points after allocation!**