Final

Short Answer

LABEL may be **too far away** for a 9-bit offset (LD). Sequence 2 works for any memory location.

ALPHA* a cannot be cast safely to **BETA* -- *a** might not be a **BETA**!

```
void applyRotation (float t, float p, ALPHA* a)
{
    BETA* b = a; // problem is here
    b->rotate3D (t, p);
}
```

subroutine A executes **JSR** without **saving R7** —infinite loop!

does not wait for display to be ready

```
        .ORIG x3000
        LD R0,NUM5
        STI R0,DDR
        HALT
NUM5    .FILL x35     ; ASCII digit '5'
DDR     .FILL xFE06
        .END
```

C++ program crashes after **main** has returned

crashes in destructor (for variable in static storage)

```
#include <stdio.h>
int weird () {
  printf ("weird");
  return 0;
}
int run () {
  char buffer[10];
  scanf ("%s", buffer);
  return 0;
}
int main() {
    run ();
    printf ("main");
    return 0;
}
```

In response, the program prints out "weird" instead of "main", then terminates. Based on your knowledge of the LC-3 calling convention

Special InputTM overwrote return address on stack with address of weird

```
int player_sort_by_rank (const void* p1, const void* p2)
{
    int32_t r1 = player_get_rank (p1);
    int32_t r2 = player_get_rank (p2);

    if (r1 > r2) { return -1; }
    if (r2 > r1) { return  1; }
    return 0;
}
```

Calculate rank once for each player and store in a new field of player_t

```
class ALPHA {
    private:
        int val;
    public:
        ALPHA (int start) : val (start) { }
        void add (int amt) { val += amt; }
        void add (double amt) { add (ceil (amt)); }
        int value (void) { return val; }
};

int
main ()
{
    ALPHA a (40);

    a.add (1.5);

    printf ("%d\n", a.value ());

    return 0;
}
```

infinite recursion to ALPHA::add with double argument

```
typedef struct book_t book_t;
struct book_t {
    // some stuff
    book_t* next;              // for the library
};

typedef struct good_book_t good_book_t;
struct good_book_t {
    book_t base;
    // some other stuff
    void (*promote_book) (void);   // a function pointer for good books
};
```

Not all books are good books!
(Not safe to cast book_t* to good_book_t*.)

```
typedef struct 3D_point_t 3D_point_t;
struct 3D_point_t {
    int32_t      x, y, z;  // coordinates of point
    double_list_t dl;      // for list of points
};
```

double_list_t field must be first in 3D_point_t!_

LC-3 Stack

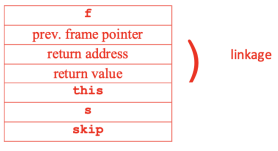
```
class ALPHA {

private:                          ALPHA::func
    char x;

public:
    ALPHA (char _x) : x (_x) { }

    char* func (const char* s, int16_t skip) {
        const char* f;

        for (f = s; '\0' != *f; ++f) {
            if (x == *f && 0 == --skip) {
                return f;
            }
        }
        return NULL;
    }
};
```



序号	函数和描述
1	void *calloc(int num, int size); 在内存中动态地分配 num 个长度为 size 的连续空间，并将每一个字节都初始化为 0。所以它的结果是分配了 num*size 个字节长度的内存空间，并且每个字节的值都是 0。
2	void free(void *address); 该函数释放 address 所指向的内存块，释放的是动态分配的内存空间。
3	void *malloc(int num); 在堆区分配一块指定大小的内存空间，用来存放数据。这块内存空间在函数执行完成后不会被初始化，它们的值是未知的。
4	void *realloc(void *address, int newsize); 该函数重新分配内存，把内存扩展到 newsize。

```
/* 动态分配内存 */
description = (char *)malloc( 30 * sizeof(char) );
if( description == NULL )
{
    fprintf(stderr, "Error - unable to allocate required memory\n");
}
else
{
    strcpy( description, "Zara ali a DPS student.");
}
```

```
// If empty list or only one element, done!
if (NULL == head || NULL == head->next) {
    return head;
}
// Otherwise, divide the list into two sublists of equal length.
divide_list (head, &fst, &sec);

// Sort each half.

fst = slow_sort (fst);

sec = slow_sort (sec);

// If fst is larger than sec, swap them (you MUST use the swap function).
if (fst->value > sec->value) {

    swap (&fst->next, &sec->next); // as shown; not needed for correctness

    swap (&fst, &sec);
}
// Reconnect fst and sec into a single list.

for (last = fst; NULL != last->next; last = last->next) { }

last->next = sec;

// Sort the rest of the list.

fst->next = slow_sort (fst->next);
// Return the sorted list.
return fst;
```

```
bird_t* find_fastest_migratory_bird (dl_t* head) {

    bird_t*   rval = NULL; // return value
    double    max = -1;    // maximum speed seen
    animal_t* a;
    bird_t*   b;

    for (dl_t* elt = head->next; head != elt; elt = elt->next) {

        a = (animal_t*)elt;

        b = (bird_t*)elt;

        if (BIRD == a->type && b->migratory && b->speed > max) {

            rval = b;

            max = b->speed;

        }
```

```
typedef struct node_t Node;
struct node_t {
    int32_t data;
    Node*   next;
};

void remove_duplicates (Node* head)
{
    if (NULL == head || NULL == head->next) {
        return;
    }
    remove_duplicates (head->next);
    if (head->data == head->next->data) {
        Node* remove = head->next;
        head->next = remove->next;
        free (remove);
    }
```

注意位置

单/双链表 Doubly-linked list/ The Link List

```
typedef struct node_t Node;
struct node_t {
    int32_t X;    // metric one: smaller is better
    int32_t Y;    // metric two: smaller is better
    Node*   next;
};
```

A node is **Pareto-dominated** if another node in the list has smaller or equal values for both X and Y.

```
void remove_dominated (Node* head)
{
    if (NULL == head || NULL == head->next) {
        return;
    }
    // original problem had non-unique X values (blue text also necessary)
    int32_t head_dom = (head->X == head->next->X && head->Y > head->next->Y);
    if (head->Y <= head->next->Y || head_dom) {
        Node* remove = head->next;
        head->next = remove->next;
        if (head_dom) {
            head->Y = remove->Y; // Note: X value is already the same.
        }
        free (remove);
        remove_dominated (head);
    } else {
        remove_dominated (head->next);
    }
}
```

List variables 静态初始化

```
static double_list_t my_list =
    {&my_list, &my_list};
```

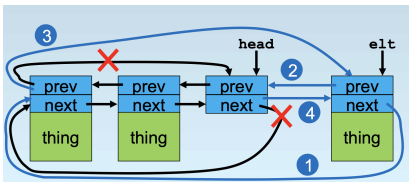
But **dynamically-allocated lists** must be **initialized at runtime.**

List variables 动态初始化

```
void dl_init (double_list_t* head)
{
    head->prev = head->next = head;
}
```

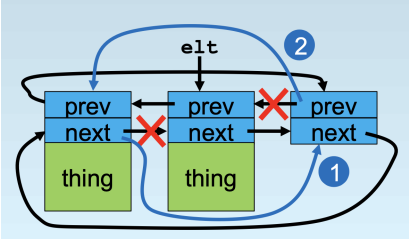
双向链表插入

```
void dl_inset (double_list_t* head, double_list_t* elt)
{
   elt->next = head->next;
   elt->prev = head;
   head->next->prev = elt;
   head->next = elt;
}
```



双向链表删除

```
elt->prev->next = elt->next; // Step 1
elt->next->prev = elt->prev; // Step 2
```



找到 List 里的第一个元素

```
void* dl_first (double_list_t* head)
{
  return (head == head->next ? NULL : head->next);
}
```

Callbacks 回调函数

```
int32_t isort (void* base, int32_t n_elts, size_t size,
               int32_t (*is_smaller) (void* t1, void* t2));
typedef struct horse_t horse_t;
struct horse_t {
    char*   name;   // dynamically allocated
    int32_t age;    // in years
    int32_t height; // in hands
};
```

int strcmp (const char* s1, const char* s2);

The **strcmp** function returns 0 iff the strings **s1** and **s2** are the same.

```
int32_t compare_horses (const void* elt1, const void* elt2)
{
    const horse_t* h1 = elt1;
    const horst_t* h2 = elt2;
    return (0 == strcmp (h1->name, h2->name) &&
            h1->age == h2->age &&
            h1->height == h2->height);
}
      // horse_t structure and compare_horses signature
      //   replicated for your convenience.
typedef struct horse_t horse_t;
struct horse_t {
    char*   name;   // dynamically allocated
    int32_t age;    // in years
    int32_t height; // in hands
};
int32_t compare_horses (const void* elt1, const void* elt2);

void* find_element (void* array, int32_t n_elts, size_t size, void* elt_to_find,
                    int32_t (*cmp) (const void* elt1, const void* elt2))
{
  char*   ar = array;
  int32_t i;

  for (i = 0; n_elts > i; i++) {
      if (cmp (elt_to_find, ar + i * size)) {
          return (ar + i * size);
      }
  }
  return NULL;
```

C++

```
class Base {
                int A;
protected:      int B;
private:        int C;
public:         int D;
};

class Derived: public Base {
private:        int E;
                static void aFunction (void);
public:         int F;
};

Derived instance;

void anotherFunction (void);
```

Derived::aFunction B D E F
anotherFunction. D F

```
#include <stdio.h>

class Mystery {
private:
    int x;
public:
    Mystery () { printf("M"); }
    Mystery (int xval) : x(xval + 1) { printf("Y"); }
    const Mystery& operator= (int xval) {
        xval = 1;
        printf("S");
        return *this;
    }
    Mystery (const Mystery& m) : Mystery(m.x + 10) { printf("T"); }
    ~Mystery() { printf("E"); }
};

Mystery c, d;

int main() {
    printf("---START---\n");
    c = d = 0;
    printf("\n");
    Mystery a = 42;
    printf("\n");
    Mystery b = a;
    printf("\n");
    c = a;
    printf("\n---END---");
    return 0;
}
```

Constructors, Destructors, and Operator Overloading

Line 1: MM---START---
Line 2: S
Line 3: Y
Line 4: YT
Line 5: blank
Line 6: ---END---EEEE

a.x = 43 b.x = 54 c.x = 43 d.x = bits

```
class Tricky {
private:
    int32_t a;
    int32_t b;
    Tricky (int32_t x, int32_t y) : a (x), b (y) { }
    friend Tricky operator& (const Tricky& t1, const Tricky& t2) {
        Tricky rval (t1.a * t2.b, t2.a * t1.b);
        return rval;
    }
    friend Tricky operator/ (const Tricky& t1, const Tricky& t2) {
        Tricky rval (t1.a / t2.a, t1.b / t2.b);
        return rval;
    }
public:
    Tricky (const Tricky& t) : a (t.a), b (t.a - 1) { }
    Tricky (double p) : a (15), b ((int32_t)round (p + 0.3)) { }
    Tricky (int32_t z) : a (z), b (z) { }

    void report (void);
};

int main ()
{
    Tricky one = 23.45;
    Tricky two = (5 & one);
    Tricky three = (one & (two / 10)) / two;

    one.report ();
    two.report ();
    three.report ();

    return 0;
}

void Tricky::report (void)
{
    printf ("%d\n", a - b);
}
```

The program's output is

____-9____
____45____
____-3____

```
MyClass* m = new MyClass
                 (arg1, arg2, ...);
```

Use **delete** to Deallocate Instances, **delete[]** for Arrays

Given **MyClass*** m,
- **delete m;** // deletes an instance
- **delete[] m;** // deletes an array

Single-Argument Constructors Create Implicit Casts

```
class complex {
    // ...
public:
    complex (int32_t real_part);
    complex (double real_part);
    friend complex operator*
        (const complex& a,
         const complex& b);
};
```

Creates implicit cast from **int32_t** to **complex**.

Creates implicit cast from **double** to **complex**.

I/O FILE

```
int32_t file_reduce (const char* fname)
{
    FILE* in;  // input stream
    FILE* out; // output stream
    // First, write code to prepare the streams for use.

    if (NULL == (in = fopen (fname, "r")) ||
        NULL == (out = fopen ("out.txt", "w"))) {
        if (NULL != in) {
            fclose (in);
        }
        return 0;
    }
// Read the input file and produce the output.

int last = EOF, char;

while (EOF != (char = fgetc (in))) {
    if (last != char) {
        fputc (char, out);
        last = char;
    }
}
    // Clean up and return.

    fclose (in);
    return (0 == fclose (out) ? 1 : 0);

}
```