ECE220 S18 Answer

Problem 1

1. The following two sequences of instructions seem to accomplish the same task, but sequence 1 uses fewer registers and fewer lines of code. Assume that the labels `LABEL` and `LATER` appear somewhere in the program.

```
; SEQUENCE 1
    LD R1, LABEL
    BRnzp LATER
; SEQUENCE 2
    LD R0, OTHER
    LDR R1, R0, #0
    BRnzp LATER
OTHER .FILL LABEL
```

   In some cases, sequence 1 may fail, while sequence 2 continues to work. USING 30 WORDS OR FEWER, EXPLAIN WHY.

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

2. Consider the following LC-3 code:

```
LOOPTOP
    ADD R1,R1,#0 ; question asks about this ADD
    BRnz NEXT_SECTION
    ADD R1,R1 #-1
    JSR DO_STUFF
    BRnzp LOOPTOP
```

   Assume that `NEXT_SECTION` and `DO_STUFF` are valid labels, and that the `DO_STUFF` subroutine does not modify R1. USING 10 WORDS OR FEWER, explain the purpose of the `ADD` instruction at the top of the loop.

   The ADD sets the condition codes for the BRnz

3. USING 30 WORDS OR FEWER, explain the problem with the code below. Be specific as to why the unacceptable code is not allowed.

```
class ALPHA {
protected:
    int x;
    int y;
};
```

```
6  class BETA : public ALPHA {
7  private:
8      int z;
9  public:
10     void rotate3D (double theta, double phi);
11 };
12 void applyRotation (float t, float p, ALPHA* a)
13 {
14     BETA* b = a; // problem is here
15     b->rotate3D (t, p);
16 }
```

`ALPHA* a` cannot be cast safely to `BETA*` because `*a` might not be a `*BETA`

4. Draw the LC-3 stack frame for the member function `ALPHA::func` shown below. Clearly label all elements of the stack frame, and show where R5 and R6 point during execution of the function's code.

```
1  class ALPHA {
2  private:
3      char x;
4  public:
5      ALPHA (char _x) : x (_x) { }
6      char* func (const char* s, int16_t skip) {
7          const char* f;
8          for (f = s; '\0' != *f; ++f) {
9              if (x == *f && 0 == --skip) {
10                 return f;
11             }
12         }
13     return NULL;
14     }
15 };
```

f <— R5, R6
caller's frame pointer
return address
return value
this
s
skip

其中的`this`是C++成员函数隐含的第一个参数。在 C++ 中，**每个非静态成员函数都会自动获得一个隐藏的参数 `this` 指针**，指向调用该函数的对象。

5. Read the following program, then write its output below.

```
#include <math.h>
#include <stdio.h>
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; // 此时调用的是Trick(double p)的构造函数，生成Tricky(15,24)
    Tricky two = (5 & one); // 此时5先被构造为Trick(5,5)，然后和Tricky(15,24)进行&操作，得到Tricky(120, 75),再调用复制函数得到Tricky(120,119)
    Tricky three = (one & (two / 10)) / two; // (two/10)得到Tricky(12,11), one&Tricky(12,11)得到Tricky(165,288)，调用复制函数得到Tricky(165,164)，再/two得到Tricky(1, 1)，再调用复制函数得到Tricky(1,0)
    one.report (); // 15-24 = -9
    two.report (); // 120-119 = 1
    three.report (); // 1-0 = 1
    return 0;
}
void Tricky::report (void){
    printf ("%d\n", a - b);
}
```

Answer: -9, 1, 1(原答案错误)

Problem 2

This problem is based on the following node structure:

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

Write a recursive function that takes one input `head`, a pointer to the head (not a sentinel) of a **sorted, singly-linked list of dynamically allocated Nodes**, and removes all duplicate elements in the list. A duplicate element is any element whose `data` field matches that of any previous element in the linked list. A solution is possible using nine lines of code.

```
void remove_duplicates (Node* head){
    if (head == NULL || head->next == NULL){return;} // 没有节点或者只有一个节点
    remove_duplicates(head->next);
    if (head->data == head->next->data){ // 如果该节点与下一个节点的data相同，则删除下一节点
        Node* remove = head->next;
        head->next = remove->next;
        free(remove);
    }
}
```

Problem 3

In lecture, we developed a generic insertion sort subroutine using the following function signature:

```
int32_t isort (void* base, int32_t n_elts, size_t size, int32_t (*is_smaller) (void* t1, void*
t2));
```

In this problem, you must develop a generic routine to find a pointer to a matching element in an array. Since you all liked my horse photos, this problem focuses on horses. The following C structure defines a horse:

```
typedef struct horse_t horse_t;
struct horse_t {
    char* name; // dynamically allocated
    int32_t age; // in years
    int32_t height; // in hands
};
```

1. Begin by writing the `compare_horses` function below, which should return 1 if the two horses are the same (all fields are the same), and 0 if they are different. You should use the standard C library routine for string comparison:`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 horse_t* h2 = elt2;
    return (strcmp(h1->name, h2->name) == 0
           && h1->age == h2->age
           && h1->height == h2->height);
}
```

2. Next, write `find_element`, which uses a callback to a function such as `compare_horses` in order to locate an element matching `elt_to_find` in an array `array` with `n_elts` elements of `size` bytes each. The function should return a pointer to the matching element in the array, or `NULL` if no such element is found.

```
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)){ // 函数指针
    char* ar = array;
    int32_t i;
    for (i = 0, i < n_elts; i++){
        if (cmp(elt_to_find, ar+i*size)){
            return (ar + i * size);
        }
    }
    return NULL;
}
```

3.) Finally, call `find_element` on the array `my_stable`, which holds 42 horses, to find the horse `my_favorite`.

```
static horse_t my_stable[42];
horse_t* h = find_element(my_stable, 42, sizeof(my_stable[0]), &my_favorite,
compare_horses);
```

Problem 4

In this problem, you must write code for objects for a game written in C++. The base class is `Obj`, but each other type of object has its own class derived from `Obj`. For simplicity, we define only one derived class: `Vehicle`.

Objects in the game are kept in a list of `Obj*` (based on the STL list template that you used in MP12). When the game is saved, the `save` function is invoked on each pointer in the list. Similarly, when the game is loaded, the `load`function is invoked on each pointer in the list.

The `save` and `load` member functions for all classes take a `FILE*` as an input and return an `int32_t`. All functions should return 0 on success, or -1 on failure.

1. Complete the class definition for the `Obj` class to support the save/load functionality just discussed. Do not include code for the functions—you must write that code in the next part.

```
class Obj {
private:
    uint64_t uid;
public:
    Obj (uint64_t _uid) : uid (_uid) { }
    virtual int32_t save(FILE* f);
    virtual int32_t load(FILE* f);
};

class Vehicle: public Obj {
private:
    char* name; // dynamically allocated using strdup; limit to 99 chars;
                // name does not contain space, tab, \n, nor \r
    int32_t type;
    double gasLvl;
public:
    Vehicle (uint64_t _uid, const char* _name, int32_t _type, double _gasLvl) :
        Obj (_uid), name (strdup (_name)), type (_type), gasLvl (_gasLvl) { }
    int32_t save (FILE* f);
    int32_t load (FILE* f);
};
```

2. Implement the `save` and `load` methods for the `Obj` class below (nothing has been given— write it all yourself). Some constraints and hints follow:
   - Do not assume that the `FILE*` argument is non-NULL, and be sure to check all return values.
   - See the reference page at the back of the exam for some of C's I/O library API.
   - Functions for specific classes can be called using the `ClassName:: prefix`.
   - The instance to which `this` points has been constructed before either function is called.

   Neither function should require more than a few lines of code. Remember that both should return 0 on success, or -1 on failure.

```
int32_t Obj::save(FILE* f){
    if (f == NULL || 0 > fprintf(f, "&ld", uid)){
        return -1;
    }
    return 0;
}

int32_t Obj::load(FILE* f){
    if (f == NULL || 1 != fscanf(f, "%ld", &uid)){
        return -1;
    }
    return 0;
}
```

3. Implement the `save` and `load` methods for the `Vehicle` class below (nothing has been given—write it all yourself). Some constraints and hints follow:

   - Do not assume that the `FILE*` argument is non-NULL, and be sure to check all return values.
   - See the reference page at the back of the exam for some of C's I/O library API.
   - Functions for specific classes can be called using the `ClassName:: prefix`.
   - The instance to which this points has been constructed before either function is called.

   Neither function should require more than a few lines of code. Remember that both should return 0 on success, or -1 on failure.

```
int32_t Vehicle::save (FILE* f){
    if (f == NULL || 0 != Obj::save (f) || 0 > fprintf(f, "%s %d %f", name, type, gasLvl))
{
        return -1;
    }
    return 0;
}

int32_t Vehicle::load (FILE* f){
    if (f == NULL || 0 != Obj::load(f)){return -1;}
    char buf[100];
    if (3 != fscanf(f, "%99s %d %f", buf, &type, &gasLvl)){return -1;}
    char* n = strdup(buf);
    if (n == NULL){return -1;}
    free(name);
    name = n;
    return 0;
}
```

4. The implementation that you have just written does not allow all objects from the list to be stored consecutively into a single file. USING 20 WORDS OR FEWER, explain the difficulty.

When loading, one cannot determine whether the next element in a file has type Obj or type Vehicle.

5. Defining the functions to load objects from a file forces these functions to work with objects that have already been constructed. To avoid this problem, write declarations below for alternative functions that can accomplish the same goal. These declarations must normally appear in the class definitions, but just write them below, showing the declarations for both Obj and Vehicle classes along with any initializers needed.

```
1  Obj (FILE* f);
2  Vehicle (FILE* f) : Obj (f);
```