COMP 2011 Final - Spring 2019 – HKUST

SOLUTION

Problem 1 String *[8 points]*

Answer:

No

No

No

Yes

Grading scheme: 2 points each

Problem 2 Lambda Expression *[5 points]*

	Answer
(a)	Compilation error
(b)	4, 5
(c)	20, 0
(d)	1.25
(e)	1

Grading scheme: 1 point each

Problem 3 Binary Tree and Queue *[8 points]*

Answer:

1 6 5 4 3 2 *Grading scheme: 3 points if the whole line of ouput is correct*

2 3 4 5 6 1 *Grading scheme: 5 points if the whole line of output is correct*

Problem 4 Stack *[7 points]*

Answer:

(a) `-c+ba` *Grading scheme: 2 points (no partial mark)*

(b) `+*cba` *Grading scheme: 2 points (no partial mark)*

(c) `-i*-e^dc+ba` *Grading scheme: 3 points (no partial mark)*

Problem 5 C++ Basics and Class Basics *[10 points]*

Line Number	Code(s) with Error	Reason/Correction
17	void Store(...)	Constructor should not have return type.
24	void setLocation(double a=0.0, ...)	Default parameter should start on the end of the parameter list.
20	Item2.name = b.name;	Array-array assignment is illegal.
27 or 28	void findTotal() const	An accessor should not modify the data member total.
41	myStore.total = 0;	The data member total is private.

Grading scheme: 2 points each unique error:

0.5 for line number, 0.5 for code, 1 pt for reason

Problem 6 Control Flow in 2D Array *[7 points]*

Answer:

1 2 3 4 8 12 16 15 14 13 9 5 6 7 11 10 (Print a given matrix in spiral form)

Grading scheme: 1 point for each underlined sequence in correct order

/* k - starting row index, m - ending row index, l - starting column index, n - ending column index, i – iterator */

Problem 7 Class *[27 points]*

a) (4 points)

Answer for the Actor class:

void printInfo(bool) const; // or

void printInfo(bool = false) const; // or

void printInfo(bool printMovies = false) const; // any parameter name is ok

Grading scheme: 2 points for the correct prototype (missing const, -0.5)

Answer for the Movie class:

void printInfo(bool = false) const; // or

void printInfo(bool printMovies = false) const; // any parameter name is ok

Grading scheme: 2 points for the correct prototype (missing const, -0.5)

Wrong parameter including missing default value: -1

b) (15 points)

```
#include "actorMovie.h"  /* File: actorMovie.cpp */ // 1 point
// Missing Class & scope operator: -2 points for the whole part (b)
Actor::Actor() { // default constructor: 1 point
   name = nullptr;   numMovies = 0;
}

Actor::Actor(const char* name){ // general constructor: 2 points
   this->name = nullptr;
   setName(name);
   numMovies= 0;
}

Actor::~Actor(){ // destructor: 1 points
   if (name != nullptr)
      delete [] name;     // 1 point
}

void Actor::setName(const char* name) { // 3 points
   if (this->name != nullptr) // to prevent memory leak
      delete [] this->name;
   this->name = new char[strlen(name)+1]; // memory allocation
   strcpy(this->name, name);
}

void Actor::addMovie(Movie *moviePtr){ // 3 points
   for (int i=0; i<numMovies; i++) // checking if movie existed
      if ((strcmp( moviePtr->getTitle(), inMovies[i]->getTitle()) == 0)
          && (moviePtr->getReleaseYear() == inMovies[i]->getReleaseYear()))
         return;
   if (numMovies < MAX_NUM) // add the pointer to the array
      inMovies[numMovies++] = moviePtr;
}

const char* Actor::getName() const { // 1 point
   return name;
}

void Actor::printInfo(bool printMovies) const { // 3 points
   cout << "\"" << name << "\"" << endl;;
   if (printMovies) {
      cout << "has played in movies:" << endl;
      for (int i=0; i<numMovies; i++)
         inMovies[i]->printInfo();
   }
}
```

c) (8 points) Implement the member function `addActorInMovie()`.

```
#include "filmStudio.h"  /* File: filmStudio.cpp  */ // 1 point

// Missing Class & scope operator: -2 points for the whole part (c)

void FilmStudio::addActorInMovie(const char* actorName, const char* movieName, int
year)
{
   int actorIndex = -1, movieIndex = -1;

   // Search the the actor  // 1 point
   for (int i=0; i<numActors; i++)
      if (strcmp(actors[i].getName(), actorName)==0)
         actorIndex = i;

   // Search for the movie // 1.5 points
   for (int i=0; i<numMovies; i++)
      if ((strcmp(movies[i].getTitle(), movieName) == 0) &&
          (movies[i].getReleaseYear() == year))
         movieIndex = i;

   // Add the actor, if it is not found and actors not full
   // 1.5 points
   if ((actorIndex == -1) && (numActors < MAX_NUM))
   {
      actorIndex = numActors;
      actors[numActors++].setName(actorName);
   }
   // Add the movie, if it is not found and movies not full
   // 1.5 points
   if ((movieIndex == -1) && (numMovies < MAX_NUM))
   {
      movieIndex = numMovies;
      movies[numMovies++].setTitleAndYear(movieName, year);
   }

   // update the inMovies & Cast relations
   // only with both actor and movie be found or added
   // 1.5 points
   if ((actorIndex > -1) && (movieIndex > -1))
   {
      actors[actorIndex].addMovie(&movies[movieIndex]);
      movies[movieIndex].addActor(&actors[actorIndex]);
   }
}
```

Problem 8 Linked List Addition [28 points]

a) (5 points) Implement the function `LL_create()`.

```
Node* LL_create(int n){
   // Your implementation starts here
    Node* head = nullptr;  // 1 point
    // pushing items: 3 points
    while (n >= 10){
       LL_push(head, n%10);
       n = n/10;
    }
    LL_push(head, n);
    return head;  // return the result: 1 point
}
```

b) (10 points) Implement the **recursive** function `LL_addSameSize()`.

```
Node* LL_addSameSize(const Node* head1, const Node* head2, int& carry){

    // memory allocation
    Node* result = new Node;
    int sum;
    // base case
    if (head1 == nullptr)
        return nullptr;
    // Recursion
    // Recursively add remaining nodes and get the carry
    result->next = LL_addSameSize(head1->next, head2->next, carry);
    // Addition
    // add digits of current nodes and propagated carry
    sum = head1->data + head2->data + carry;
    carry = sum / 10;
    sum = sum % 10;
    // Store the sum
    // Assign the sum to current node of resultant list
    result->data = sum;
    // return the new linked list
    return result;
}
```

c) (13 points) Implement the function `LL_add()`.

```
// Declaration of helper function
void addCarrytoRemaining(const Node* head1, const Node* cutpoint, int&
carry, Node*& result);

void LL_add(const Node* head1, const Node* head2, Node*& result){
// Your implementation starts here
    // see grading scheme at the end
    int size1 = LL_size(head1);
    int size2 = LL_size(head2);
    int diff = abs(size1 - size2);
    int carry = 0;
    const Node* cutpoint = nullptr;
    if (size1 == size2){ // two lists of same size
        result = LL_addSameSize(head1, head2, carry);
    }
    else { // two lists of different size
        if (size1 < size2){ //set list1 be the longer list
            const Node* temp = head1;
            head1 = head2;
            head2 = temp;
        }
        for (cutpoint = head1; diff--; cutpoint = cutpoint->next);
        result = LL_addSameSize(cutpoint, head2, carry);
        addCarrytoRemaining(head1, cutpoint, carry, result);
    }
    if (carry)
        LL_push(result, carry);
}

// Implementation of helper function
void addCarrytoRemaining(const Node* head1, const Node* cutpoint, int&
carry, Node*& result){
    int sum;
    if (head1 == cutpoint)
        return;
    else{
        addCarrytoRemaining(head1->next, cutpoint, carry, result);
```

```
            sum = head1->data + carry;
        carry = sum/10;
        sum %= 10;
        LL_push(result, sum);
    }
}
// grading scheme:
// alignment: 4 points
// addition: 3 points
// carry: 4 points
// create new linked list: 2 points
```

~ End of Paper ~