CSP2348/CSP5243 Data Structures

Tutorial 05: SLL Practice

Related Objectives from Unit Outline:

- Describe (arrays, linked lists, binary trees, and hash tables) data structures and analyse the complexity and performance of their associated algorithms.

Objectives:

1. To become familiar with the general properties of lists;
2. To become familiar with the specific properties of list arrays;
3. To demonstrate the awareness of the principles of algorithms in list insertion, deletion, searching, and merging.
4. To become familiar with the tasks using SLL for the Assignment 2

Tasks:
Complete the following

Task 1: Catch up tutorial activities in Module 04

Task 2: Java practice using SLL.

During the lab session, your tutor will demonstrate the execution of a Java code, Unit_List.java, to demonstrate how to build an SLL. This example represents a list of semester results of a unit. Each SLL node contains a record of one student's results of the semester, that is, it stores a student ID followed by marks of three assessment components (A1_result, A2_result, and exam.result). For simplicity, all marks are integers. It also shows the technique of traversing an SLL, searching an SLL and inserting a node into the SLL.

This helps understanding of the linked list data structure and concepts and operations, etc. (attached below is the code, which was tested on BlueJ)

```
/**
 * It demonstrates how to build an SLL, which in this example represents a list of semester
 * results of a unit. Each SLL node contains a record of one student's results of the semester,
 * that is, it stores a student ID followed by marks of three assessment components
 * (A1_result, A2_result, and exam_result). For simplicity, all marks are integers. It also
 * shows the technique of traversing an SLL, searching an SLL and inserting a node into the
SLL.
 * Hope this helps  understanding of  the linked list data structure and concepts and
operations, etc..
 * (Code tested on BlueJ)
 *
 * @author j XIAO
```

```
 * @version v2
 */
public class unit_list
{
  private int student_ID;     //of 4 digits
  private int A1_result;      //0<= a1_mark<=20
  private int A2_result;      //0<= a2_mark<=30
  private int exam_result;    //0<= a3_mark<=50
  private unit_list next =null;
 private unit_list(int ID, int mark1, int mark2, int mark3)
  { if ((ID<999)||(ID>9999)) return;
    if ((mark1 <0.0)||(mark1>20.0)) return;
    if ((mark2 <0.0)||(mark2>30.0)) return;
    if ((mark3 <0.0)||(mark1>50.0)) return;
    this.student_ID=ID;
    this.A1_result=mark1;
    this.A2_result=mark2;
    this.exam_result=mark3;
  }
private static void highest_result(unit_list u_list)
{ // serach student with hishest overall result, of mark1+mark2+mark3
  if (u_list== null)  return;

  unit_list highest_mark = u_list;
  for (unit_list curr= u_list.next; curr!= null; curr = curr.next)
      if (curr.A1_result + curr.A2_result+curr.exam_result >
          highest_mark.A1_result + highest_mark.A2_result+highest_mark.exam_result)
highest_mark = curr;

  System.out.println("\nstudent with highest overall results is the one with Student_No.:
"+highest_mark.student_ID);
}

private static void print_unit_result(unit_list u_list)
{if (u_list == null) return;
 for (unit_list curr= u_list;  curr!= null; curr = curr.next)
  System.out.println("\nStudent_No.: "+curr.student_ID +
                "\n   A1_mark: "+curr.A1_result +
                "\n   a2_mark: "+curr.A2_result +
                "\n   Exam_mark: "+curr.exam_result);
}

/*************************INSERT*****************/

private static void insert_unit_result(unit_list u_list, int ID, int mark1, int mark2, int mark3)
{
```

```
unit_list new_node = new unit_list(ID, mark1,  mark2, mark3);
// if empty list, insert as the only node
if (u_list == null)
  return;            // cannot insert anyway due to Void return - so we assume unit_list != null

// For convenience,student records are listed in ascending order by the student_ID field.

unit_list previous = null;
unit_list curr = u_list;

while (curr!=null)                    //traverse the SLL
   { if (curr.student_ID >=ID) break;      //insert here??
     previous=curr;
     curr=curr.next;
   };
if (curr==null) /* insert as the last */
   {previous.next=new_node;
     return;
   }

if (curr.student_ID ==ID)     // ID match, replace the unit marks
  {curr.A1_result = mark1;
   curr.A2_result = mark2;
   curr.exam_result = mark3;
   return;
}
if (previous==null)  /*the new node to be inserted at the beginning */
   {new_node.next=u_list;
      // due to void return, changing unit_list link would not work
     unit_list temp = new unit_list (0, 0, 0,0);
     temp.student_ID =u_list.student_ID;
     temp.A1_result = u_list.A1_result;
     temp.A2_result = u_list.A2_result;
     temp.exam_result = u_list.exam_result;

     temp.next=u_list.next;

     u_list.student_ID =ID;
     u_list.A1_result = mark1;
     u_list.A2_result = mark2;
     u_list.exam_result = mark3;
     u_list.next = temp;
     return;
   }
// Otherwise i.e., curr.ID >ID and Previous!=null

     new_node.next=curr;
```

```
    previous.next=new_node;
    return;
}


public static void main(String[] args)
{int[] unit1 = {1111, 17, 22, 30,
      1112, 10, 6,  50,
      1114, 14, 21, 30,
      1116, 8,  16, 35,
      1122, 11, 19, 40,
      1145, 9,  16, 20,
      1189, 20, 30, 50};

//build a link of a unit result

 //first unit node
 unit_list u_list = new unit_list( unit1[0], unit1[1],  unit1[2], unit1[3]);
 unit_list curr = u_list;
 for (int i=1; i<=6; i++)    // to build the rest of the list
  {unit_list one_node = new unit_list(unit1[i*4],   //student_ID
                              unit1[i*4+1],   //a1_mark
                              unit1[i*4+2],   //a2_mark
                              unit1[i*4+3]    //exam_mark
               );
   curr.next = one_node;
   curr= curr.next;
  }

 //print out the student results of unit 1
  print_unit_result (u_list);
 //fing hishest performance student
  highest_result(u_list);

  insert_unit_result(u_list, 1225, 17, 20, 20);

   print_unit_result (u_list);
  }
}
```