任课教师______ 姓名______ 学号______ 院系______

装 订 线

浙江大学 2013－2014 学年夏季学期

《C 程序设计专题》课程期末考试题参考答案

课程号：__211Z0050__， 开课学院：__计算机学院__

考试试卷：√A 卷、B 卷（请在选定项上打 √）

考试形式：√闭、开卷（请在选定项上打 √），允许带__/__入场

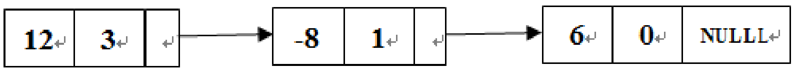
考试日期：__2014__年__06__月__27__日，考试时间：__120__分钟

试题号	一	二	三	四	总分	
满分	26	30	24	20		
得分					统分人 1	
阅卷人					统分人 2	

Section 1: Single Choice(2 marks for each item, total 26 marks)

1 C	2 D	3 B	4 A	5 C
6 C	7 D	8 D	9 D	10 D
11 C	12 **A**	13 C		

Section 2: Read the following problems and answer questions (5 marks for each item, total 30 marks)

1. 

2. 1->2->2->4->3->5

3. 186

4. void (*p)(int x, int *pa[ ]);

5. Answer:

The compiler takes printf as a function that returns int and take integers as input. The "hello\n" happens to be a pointer that is as long as an integer. Then the linker figures out how to link the call with the function.

If the function to be called takes something other than int, say double, as input. And the caller put int as the real paramter to that function. Without header file, in which should be the prototype of the function to be called, it compiles. But during run-time, the function gets an integer, while it needs a double. That will be a run-time error.

6. Answer:

线性查找算法：从第 1 个数据开始，依次比较每个元素是否是要查找的元素，直至找到；或到达数据集的末尾，确定数据集中没有要找的数据。

二分查找算法：每次与排好序的数据集的中间位置元素进行比较，根据比较结果确定是否找到，或继续在剩下的一半数据集中，继续按照上面的原则进行比较，直到找到，或确定数据集中没有要找的数据。

对于 N 个数据的查找问题：

线性查找算法的效率是比较 N 次

二分查找算法是比较 $\log_2 N$ 次

线性查找算法适用数据量小的情况，对数据没有特别要求；二分查找算法要求数据集是排好序的。

Section 3: According to the specification, complete each program (2 marks for each blank, total 24 marks)

(1)	NULL	(2)	pre->next
(3)	result_head.next	(4)	l1->val < l2->val
(5)	l2->next	(6)	l1 ? l1 : l2 or l2 ? l2 : l1
(7)	canceltimer(timeID)	(8)	timerCallBack
(9)	0, 1000	(10)	left, last-1, comp
(11)	last+1, right, comp	(12)	str, 0, 2, numcomp

院系＿＿＿＿＿ 学号＿＿＿＿＿ 姓名＿＿＿＿＿ 任课教师＿＿＿＿＿

装……订……线

Section 4: Algorithms design (10 marks for each item, total 20 marks)

1. Answer:
(1) ***Basic idea***:

Suppose that the linked list has ***M*** nodes, and let pointers ***p*** and ***q*** point both to the head of the linked list.

First, move the pointer ***q*** to the ***n***th node from the head, then there are ***M-n*** nodes left in the list.

Second, continue to move ***q*** to the end of the list, while another pointer ***p*** move at the same time, to the node ***M-n*** form the head, or to the ***n***th node from the end of the list.

Third, delete the node ***p*** points to.

Algorithm:

Let: a temporary node ***dummy***, three node pinters ***p***, ***q***, and ***tmp***;

Insert the ***dummy*** node previous to the head of the list : ***dummy.next =head;***

and let ***p*** and ***q*** point both to the ***dummy*** node: ***p = q = &dummy;***

loop ***n*** times, let ***p*** point from the dummy node to the ***n***th node from the ***dummy*** node;

continue to loop ***M+1-n*** times, move ***q*** which points to the ***n***th node from the ***dummy*** node to the end of the list, while move the pointer ***p*** which points to the ***dummy*** node to the ***M+1-n***th node from the ***dummy*** node, or the ***n+1***th node form the end of the list.

Delete the node next to the node ***p*** points to:

tmp=p-> next;   p->next=p->next->next; free(tmp);

Return the node pointer which the ***dummy*** node next to: ***return dummy.next***

(2) ***The function definition***:

```
ListNode *removeNthFromEnd(ListNode    *head,    int    n)
{
     ListNode dummy;
      ListNode *p = &dummy, *q = &dummy;
     ListNode *tmp;
     int i;

    dummy.next = head;
    for (i = 0; i < n; i++)    q = q->next;
    while (q->next) {
          p = p->next;
          q = q->next;
    }
    tmp = p->next;
    p->next = p->next->next;
    free(tmp);
    return dummy.next;
}
```

2.The function definition:

```
static void ListMnemonics(string str)
{
       string ptr;

       if (*str=='\0') { /*字符串为空，则表明本次助记符结束*/
              Mnemonics[index] = '\0'; /*形成字符串*/
             printf("%s\n", Mnemonics); /*打印*/
              return; /*返回*/
       }

       ptr = dialpad[*str-'0']; /*第一个号码数字所对应的字母字符串指针*/
       while (*ptr != '\0') {    /*对本号码数字对应的所有字母分别处理*/
               Mnemonics[index++] = *ptr; /*记录当前字母*/
                ListMnemonics(str+1); /*递归显示从下一个号码数字开始助记符*/
               index--;                      /*退回到前一个助记符位置-重要！ */
               ptr++;                         /*移到本号码数字对应的下一个字母*/
       }
       return;
   }
```