國立臺灣大學99學年度碩士班招生考試試題

科目：軟體設計

1. (15%) We wrote a recursive function `fab` to compute Fibonacci numbers, and we used a global array `count` to store the number of times `fab(n)` was called. Please predict the output of this program, and **give detailed reasoning for your answers.**

```
#include <stdio.h>
#define N 10

int count[N + 1];

int fab(int n)
{
  /* Keep track the number of times fab(n) was called */

  count[n]++;

  if (n == 0 || n == 1)
    return 1;
  else
    return(fab(n - 1) + fab(n - 2));
}

int main()
{
  int i;

  /* Although global variables are initialized automatically, we still
  do this to avoid confusion. */

  for (i = 0; i <= N; i++)
    count[i] = 0;

  fab(N);

  /* Print the values of count[]. */

  for (i = 0; i <= N; i++)
    printf("count[%d] = %d\n", i, count[i]);
}
```

2. (15%) We consider a special queue that supports three operations – insert, delete, and peek. The insertion and deletion are the same as for an ordinary queue – the insertion inserts an element at the tail of the queue, and the deletion deletes an element from the head of the queue. However, the special peek operation will return the value of

見背面

the element at the head of the queue *without* deleting it. For example, if a queue is empty and we insert 1 into a queue, then insert 2 into the queue, then peek into the queue, then delete an element from the queue, then both the results of the peek and the deletion will be 1, and 2 will remain in the queue.

We would like to write a program to output numbers from a triangle that is similar to Pascal's triangle, except that we add a 0 at the beginning of every row of Pascal's triangle. Figure 1 illustrate this triangle. One thing nice about this triangle is that, similar to the original Pascal's triangle, the value of an element, if it is not the first or the last element in a row, is still the sum of the two elements in the row above.

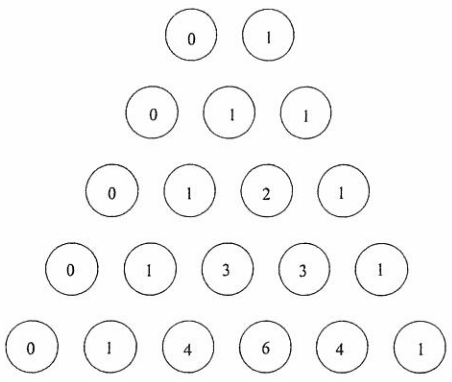

Figure 1: A triangle similar to Pascal's triangle

We now have a program that outputs the numbers from Figure 1 using this special queue. The numbers will be printed from top to bottom, and from left to right. Please fill in the blanks in the code and write down the entire loop body as your answer. Each place should be a *variable*, a *constant*, or a *function call*. Note that in each iteration we need to print x as an element in the output sequence.

You must place correct answers in *all* blanks to receive points for this problem.

```
#include <stdio.h>

/* we use a global variable to simplify the code */

#define N 1000 /* 1000 is sufficient for our purpose */
struct Queue {
  int element[N];
  int head, tail;
} queue;

void init(void)
```

接次頁

```
{
  queue.head = 0;
  queue.tail = 0;
}

void insert(int i)
{
  /* to simplify the code no checking for a full queue here */
  queue.element[queue.tail++] = i;
}

int delete(void)
{
  /* to simplify the code no checking for an empty queue here */
  return(queue.element[queue.head++]);
}

int peek(void)
{
  /* to simplify the code no checking for an empty queue here */
  return(queue.element[queue.head]);
}

int main()
{
  int x, i;

  init();
  insert(0);
  insert(1);

  for (i = 0; i < 20; i++) {
    if ((_____ = _____) == _____)
      _____;
    printf("%d\n", x);
    _____;
  }
}
```

3. (20%) A binary heap is a binary tree where the key of a node is no less than the keys of its children. As a result the root of the tree has the *largest* key. To simplify the implementation within an array we also enforce that the binary tree is *complete* – that is, every level, except possibly the last, is completely filled, and all nodes are as far left as possible. Figure 2 illustrates an binary heap.

見背面

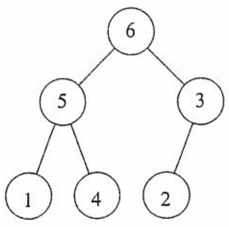
Figure 2: A binary heap

A d-heap is a just like a binary heap, except that now a node can have up to d children. Also for ease of implementation within an array we also enforce that the tree is complete – that is, every level, except possibly the last, is completely filled, and all nodes are as far left as possible. Figure 3 illustrates an d-heap when d is 4.

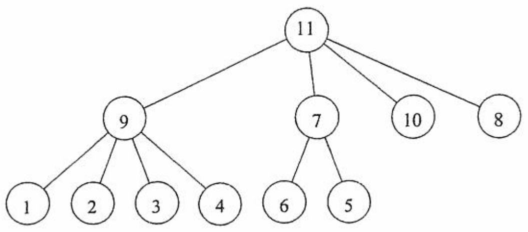
Figure 3: A 4-heap

A heap sort uses a d-heap to sort keys in two steps. The first step is to build a d-heap, and the second step is to repeatedly remove the largest key from the root of the d-heap in order to form a sorted sequence (from the largest to the smallest key).

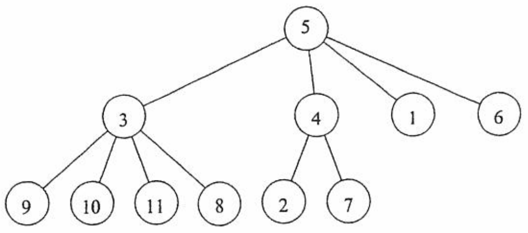
Figure 4: The initial configuration

(a) Please derive algorithms for building a d-heap in the first step, and repeatedly removing the largest element from the d-heap in the second step. **DO NOT write any pseudo code or program – they will be ignored.** Instead you

接次頁

need to explain the **key ideas** in the algorithms, and use the example in Figure 4 to illustrate how your algorithms work. (10%)

(b) Please analyze the time complexity of the algorithms in the worst case. In particular please analyze the impact of d on the complexity of the two algorithms for building heap and removing the maximum elements. (10%)

4. (15%) Given is a chain $\langle A_1, A_2, A_3, A_4, A_5 \rangle$ of five matrices whose dimensions are 40×20, 20×50, 50×30, 30×60, and 60×40, respectively. You are asked to fully parenthesize the product $A_1A_2A_3A_4A_5$ in a way that *minimizes* the number of scalar multiplications.

   (a) What is your parenthesization? (10%)

   (b) What is the number of its scalar multiplications? (5%)

5. (15%) Give a pseudocode that determines whether or not an undirected graph $G = (V, E)$ contains a cycle. You need to show that your pseudocode runs in $O(V)$ time. You may assume that the adjacency-list representation of G is given, and for each vertex u in V, $Adj[u]$ consists of all the vertices adjacent to u in G.

6. (20%) Given a universal set $\mathcal{U}$ and a collection $\mathcal{C} = \{C_1, C_2, ..., C_n\}$ of subsets of $\mathcal{U}$, the set-covering problem is to find a minimum-size subcollection of $\mathcal{C}$ that covers all elements of $\mathcal{U}$. The SET-COVER-GREEDY algorithm iteratively picks the set that covers the most remaining uncovered elements until all elements are covered.

   (a) Complete the following pseudocode by filling $\boxed{\text{6A}}$ and $\boxed{\text{6B}}$. (10%)

```
Algorithm: SET-COVER-GREEDY($\mathcal{U}, \mathcal{C}$)
1  $\mathcal{R} \leftarrow \mathcal{U}$
2  $\mathcal{C}' \leftarrow \phi$
3  while $\mathcal{R} \neq \phi$ do
4      Select a set $C_i$ from $\mathcal{C}$ that maximizes $\boxed{\text{6A}}$
5      $\mathcal{C}' \leftarrow \mathcal{C}' \cup \boxed{\text{6B}}$
6      $R \leftarrow \mathcal{R} - C_i$
7  endwhile
8  return $\mathcal{C}'$
```

   (b) Give an example for which SET-COVER-GREEDY yields a solution that is double the size of its minimum-size set cover. (10%)

試題隨卷繳回