Sinh viên PHẢI nộp lại đề thi

MSSV: ..... **Họ tên:** .....

ĐẠI HỌC BÁCH KHOA TP.HCM
Khoa Khoa học & Kỹ thuật Máy tính

KIỂM TRA GIỮA KỲ
Cấu trúc dữ liệu & Giải thuật (2012 - 2013)

Mã đề thi: 0001
Thời gian làm bài: 75 phút
Sinh viên ĐƯỢC sử dụng tài liệu, nhưng KHÔNG ĐƯỢC sử dụng Máy tính xách tay

Hướng dẫn làm bài:
Sinh viên phải điền mã số và họ tên vào các tờ đề thi
Phần I: TRẮC NGHIỆM. Sinh viên làm trên đề thi bằng cách khoanh tròn câu trả lời đúng nhất
Phần II: ĐIỀN CHỖ TRỐNG. Sinh viên làm trên đề thi
Phần III: VIẾT CHƯƠNG TRÌNH. Sinh viên làm trên giấy thi

PHẦN I: TRẮC NGHIỆM (3.0 điểm)

Để được trọn số điểm phần này, sinh viên lớp thường chỉ cần trả lời đúng 7 câu; sinh viên lớp KSTN chỉ cần trả lời đúng 9 câu.

1) Giả sử dùng mảng *A*[*m*] để hiện thực hàng đợi vòng (*circular queue*), vị trí của đầu và cuối được xác định bởi *front* và *rear*, số lượng phần tử hiện có trong hàng đợi là

A. `(rear-front+m)%m` + 1
B. `rear-front`
C. `(front-rear+m)%m`
D. `(rear-front)%m`

2) Giá trị của lời gọi hàm `recur(2)` bằng bao nhiêu?

```
int recur (int n){
 if (n==0)
  return 0;
 else
  return recur(n-1) + n*n*n;
}
```

A. 0 B. 3
C. 9 D. Các lựa chọn khác đều sai

3) Trong danh sách liên kết kép, giả sử p trỏ vào một nút trong danh sách, thao tác chèn nút q trước nút p là

A. `p→previous = q; q→next = p;`
`p→previous→next=q; q→previous=q;`
B. `p→previous=q; p→previous→next=q;`
`q→next=p; q→previous=p→previous;`
C. `q→next=p; q→previous=p→previous;`
`p→previous→next=q; p→previous=q;`
D. `q→previous=p→previous; q→next=q;`
`p→previous=q; p→previous=q;`

4) Trường hợp xấu nhất (worst case) của việc tìm kiếm tuần tự đối với mảng xảy ra khi

A. Phần tử tìm kiếm nằm ở khoảng giữa mảng
B. Phần tử tìm kiếm không nằm ở trong mảng
C. Phần tử tìm kiếm nằm ở cuối mảng
D. Cả B và C đều đúng.

5) Nhận định nào sau đây KHÔNG ĐÚNG về đệ quy vô hạn

A. Đệ quy vô hạn khiến cho chương trình bị treo.
B. Đệ quy vô hạn tiêu tốn toàn bộ bộ nhớ của hệ thống dành cho chương trình và khiến cho chương trình kết thúc một cách bất thường.
C. Gọi đệ quy gián tiếp luôn gây ra đệ quy vô hạn.
D. Nếu lời gọi đệ quy không đi đến điểm dừng (base case) thì đệ quy vô hạn sẽ xuất hiện.

6) Tìm độ phức tạp của giải thuật sau:

```
for(int i = 1; i < n; i++)
 for(int j = n; j > i; j--)
  for(int k = 1; k < n; k *= 2)
   //something that has O(1)
```

A. $O(n \log n^2)$ B. $O(n^2 \log n)$
C. $O(n^3)$ D. $O(n^2)$

7) Đưa n giá trị theo thứ tự a_1, a_2, a_3, …, a_n vào trong ngăn xếp, sau đó lần lượt lấy n giá trị này ra khỏi ngăn xếp. Nếu phần tử đầu tiên

lấy ra khỏi ngăn xếp là a_n, thì phần tử thứ i (1 ≤ i ≤ n) lấy ra khỏi ngăn xếp sẽ là

A. Không xác định B. a_{n-i+1}

C. a_i D. a_{n-i}

8) Đối với hàng đợi, việc thêm và lấy dữ liệu tuân thủ nguyên tắc

A. Vào trước ra trước

B. Vào sau ra trước

C. Vào trước ra sau

D. Không có trình tự

9) Tìm độ phức tạp của giải thuật sau:

```
i=s=0;
while(s<n)
{  i++;
   s+=i;
}
```

A. O(n) B. $O(\log_2 n)$

C. O(i) D. $O(\sqrt{n})$

10) Giả sử ngăn xếp không rỗng, thao tác nào sau đây, không làm thay đổi ngăn xếp

A. push B. pop

C. top D. Cả A, B, C

11) Cho biết phát biểu nào sai

A. Danh sách liên tục (contiguous list) chiếm một vùng nhớ liên tục

B. Danh sách liên tục thuận tiện cho thao tác chèn và xóa phần tử

C. Danh sách liên kết (linked list) không nhất thiết phải chiếm một vùng bộ nhớ liên tục

D. Danh sách liên kết thuận tiện cho thao tác chèn và xóa phần

PHẦN II: ĐIỀN CHỖ TRỐNG (3.0 điểm)

Để được trọn số điểm phần này, sinh viên lớp thường chỉ cần trả lời đúng 6 câu; sinh viên lớp KSTN chỉ cần trả lời đúng 8 câu.

1) Giá trị của biểu thức tiền tố (*infix*): - + * 9 + 3 18 * + 2 9 7 3 bằng _____263_____.

2) Xét một mảng có kích thước bằng 4 để hiện thực hàng đợi vòng, giả sử hiện tại giá trị của `rear` và `front` lần lượt là 0 và 3. Nếu xóa 1 phần tử khỏi hàng đợi, rồi thêm vào hàng đợi 2 phần tử nữa, lúc đó giá trị của `rear` và `front` lần lượt là ___2_____ và ____0____.

3) Trong một danh sách liên kết kép, khi xóa một nút được tham khảo bởi con trỏ *p*, cần cập nhật con trỏ này như sau:

`p→previous→next` = _p->next_____ ; __p->next->previous____= `p→previous`;

4) Trong một danh sách liên tục có *n* phần tử, việc xóa phần tử thứ *i* (0≤ *i* ≤ *n*-1) cần phải di chuyển _____n-i-1_____ phần tử.

5) Biểu thức *a**(*b* + *c*) – *d* biểu diễn dưới dạng hậu tố là ____a b c + * d -_____.

6) Xét một danh sách liên tục có thứ tự và gồm *n* phần tử.

Độ phức tạp của giải thuật tìm kiếm tuần tự là _____O(n)_____,

Độ phức tạp của giải thuật tìm kiếm nhị phân là _____O(logn)_____.

7) Cho đoạn mã sau, giả sử thời gian thực hiện câu lệnh A (**Statement A)** lớn hơn nhiều so với các câu lệnh còn lại

```
i = n;
while (i >= n){
  j = 1;
```

```
        while (j <= i){
          Statement A;
          j = j + 1;
        }
        i = i - 1;
      }
```

Số lần câu lệnh A được thực hiện là ____n lần_____, từ đó có thể suy ra độ phức tạp của giải thuật là _____O(An)_____.

8) Giả sử trong ngăn xếp có *n* phần tử, nếu tiến hành thêm dữ liệu thì ngăn xếp bị tràn trên (*overflow*). Điều này chứng tỏ dung lượng lớn nhất của ngăn xếp là _____n_____.

9) Cho hàm đệ quy

```
int f(int x){
  return ((x > 0) ? x*f(x-1): 2);
}
```

Giá trị của `f(f(1))` = f(1*f(0)) = f(f(0)) = f(2) = 2*f(1) = 2*1*f(0) = 2*1*2 = 4

10) Chức năng của hàm đệ quy sau là tính _____x^n_____

```
float foo(float x, int n){
   if(n == 0)
     return 1;
   else
     return x*foo(x, n-1);
}
```

PHẦN III: VIẾT CHƯƠNG TRÌNH (4 điểm)

Câu 1: (2 điểm)

Hiện thực hàm **`convert`** sau đây nhằm chuyển đổi số hệ thập phân sang số hệ bát phân.

```
void convert(int num, char*& oct){
      //Phần hiện thực của sinh viên
}
```

Lưu ý:

- **`num`** là một số nguyên dương;
- **`oct`** là chuỗi các ký tự trong tập số hệ bát phân {'0', '1', '2', '3', '4', '5', '6', '7'}. Ký tự kết thúc chuỗi luôn là '\0', sinh viên phải tự chèn vào trước khi trả về. Giả sử bộ nhớ được tham khảo bởi **`oct`** là đủ lớn và đã được cấp phát trước khi gọi hàm.
- Sinh viên được khai báo thêm *chỉ một stack trung gian* và các biến thuộc kiểu cơ bản khác nếu cần. Tuy nhiên, sinh viên không được khai báo thêm biến phụ khác thuộc kiểu danh sách.
- class Stack đã được hiện thực sẵn, sinh viên chỉ dùng; ý nghĩa các phương thức của stack được định nghĩa trong phần lý thuyết.

```
class stack {
   private:
        int data[];
   public:
        bool pop(int &n);
        bool push(int n);
        bool top();
        bool isFull();
        bool isEmpty();
        bool size();
};
```

Solution:

```
void convert(int num, char* &oct)
{
	int temp;
	stack s = new stack();
	char* ss = new char();
	while(true)
	{
		temp = num%8;
		s.push(temp);
		num = num/8;
		if(num==0)
		{
			break;
		}
	}
	while(! s.isEmpty())
	{
		s.pop(temp);
		sprintf(ss,"%d",temp);
		strcat(oct,ss);
	}
	strcat(oct,"\0");
}
```

Sinh viên lớp thường thì làm Câu 2.A, còn sinh viên lớp "Tài năng" làm Câu 2.B sau đây

Câu 2.A (2 điểm): của lớp Thường

Nút trong một danh sách liên kết được định nghĩa như sau:

```
struct node {
  int   data;
  node  *next;
};
```

Hãy viết hàm **RedundantProcessing** như sau để xử lý những dữ liệu trùng lặp trong một danh sách liên kết; tức là trong trường hợp có hai hoặc nhiều nút có giá trị dữ liệu giống nhau, thì chỉ giữ lại nút đầu tiên, các nút còn lại sẽ được đưa về cuối danh sách. Hàm **RedundantProcessing** trả về một số nguyên cho biết số phần tử bị trùng và bị đưa về cuối danh sách:

```
int RedundantProcessing (node*& head){
   //Phần hiện thực của sinh viên
}
```

Ví dụ: Giả sử có danh sách A như sau:

A: 10 15 8 **10** 20 **8** **15** 13

Sau khi gọi hàm **RedundantProcessing,** danh sách A trở thành như sau và hàm trả về 3

A: 10 15 8 20 13 **10** **15** **8**

Solution:

```
int RedundantProcessing(node* &head){
	int count;
	int data;
	node* temp1, temp2, preTemp2, redundant, tempRedun;
	if(head == NULL || head->next = NULL)
	    return 0;
	temp1=head;
	while(temp1->next != NULL){
	    data = temp1->data;
```

```
            temp2=temp1->next;
            preTemp2 = temp1;
            while(temp2!= NULL){
                  if(temp2->data == data){
                        count++;
                        if(count == 1){
                              redundant = temp2;
                              tempRedun = redundant;
                        }
                        else{
                              tempRedun->next = temp2;
                        }
                        preTemp2->next = temp2->next;
                        temp2 = temp2->next;
                        tempRudun->next = NULL;
                  }
                  else{
                        preTemp2 = temp2;
                        temp2 = temp2->next;
                  }
            }
            temp1=temp1->next;
      }
      temp1->next = redundant;
      return count;
}
```

Câu 2.B (2 điểm): của lớp Tài năng

Nút trong một danh sách liên kết được định nghĩa như sau:

```
struct node {
   int  data;
   node *next;
};
```

Hãy viết hàm `mergeList` để trộn 2 danh sách liên kết đã có thứ tự và trả về một danh sách cũng có thứ tự. Giả thiết rằng thứ tự là không giảm.

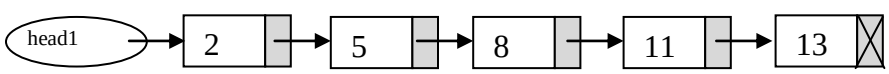
Lưu ý:

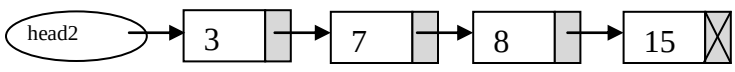
- Sinh viên **KHÔNG** được sử dụng tác vụ để tạo node mới (**nghĩa là: new node()**) trong hàm `mergeList`. Sinh viên chỉ thay đổi con trỏ để tạo ra danh sách kết quả.
- Hàm `mergeList` được cho như sau. `head1` và `head2` là con trỏ đến 2 danh sách đầu vào; `head3` là con trỏ đến danh sách trả về. Hàm `mergeList` trả về số phần tử trong danh sách `head3`.

```
int mergeList(node* head1, node* head2, node*& head3){
      //Phần hiện thực của sinh viên
}
```

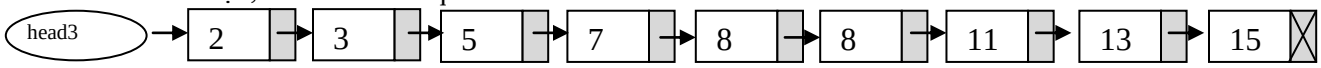
Ví dụ:

Giả sử có 2 danh sách như sau:





Sau khi trộn, danh sách kết quả là:



Solution:

```
int mergeList(node* head1, node* head2, node*& head3){
	node* temp;
	if(head1 == NULL){
		head3 = head2;
		return 0;
	}
	if(head2 == NULL){
		head3 = head1;
		return 0;
	}
	if(head1->data >= head2->data){
		head3 = head1;
		head1 = head1->next;
	}
	else{
		head3 = head2;
		head2 = head2->next;
	}
	temp = head3;
	while(head1 != NULL && head2 != NULL){
		if(head1->data >= head2->data){
			temp->next = head1;
			head1=head1->next;
		}
		else{
			temp->next = head2;
			head2 = head2->next;
		}
		temp = temp->next;
	}
	if(head1 == NULL && head2 != NULL){
		temp->next = head2;
	}
	else if(head2 == NULL && head1 != NULL){
		temp->next = head1;
	}
	return 0;
}
```

---------- HẾT ----------