計算機程式設計 113 學年度第 2 學期小考 4 試題

系級：_____ 學號：_____ 姓名：_____

All programs have "#include <stdio.h>".

1. Comparing "Linked List" with "Array". Fill the best one of the following: **(A)** Variable(s), **(B)** Link list(s), **(C)** Main, **(D)** Array(s), **(E)** Function(s), **(F)** Abstraction. **(16 %)**

(1) __D__ is a static data structure.

(2) __B__ is a dynamic data structure.

(3) __D__ store elements in contiguous memory locations, and allow faster access to an element at a specific index.

(4) __B__ are less rigid in their storage structure and elements are usually not stored in contiguous locations.

(5) __B__ need to be stored with additional memory giving a pointer to the next element.

(6) __B__ allow insertion and deletion in the middle in **O(1)** time.

(7) For implementation of Queue and Deque, simple__D__ (not circular queue) implementation is not efficient at all.

(8) __B__ is easy and straightforward for implementation of Queue and Deque.

(9) __B__ may be more space efficient in cases where it cannot be guessed the required number of elements.

(10) __B__, it is O(n) search operation due to sequential access.

2. Fill in the space to complete the code. **(10%)**

(1) _-1_ (2) _-1_ (3) _++(*p_top) / ++*p_top_ (4) _(*p_top)--__

Output	Push success=1, top=0 Push success=1, top=1 Pop success=1, data=8, top=0 Pop success=1, data=5, top=-1 Pop success=0, data=5, top=-1

```
01 #define MaxSize 7
02 int isEmpty(int top) { return top == _____(1)_____; }
03 int isFull(int top) {    return top == MaxSize_____(2)_____;  }
04 int push(int stack[MaxSize], int *p_top, int n) {
05     if (!isFull(*p_top)) {
06         stack[_____(3)_____] = n;
07         return 1;
08     }
09     return 0;
10 }
11 int pop(int stack[MaxSize], int *p_top, int *data) {
12     if (!isEmpty(*p_top)) {
13         (*data) = stack[_____(4)_____];
14         return 1;
15     }
16     return 0;
17 }
18 int main() {
19     int stack[MaxSize];
20     int top = -1, flag, data;
21     flag = push(stack, &top, 5);
22     printf("Push success=%d, top=%d\n", flag, top);
23     flag = push(stack, &top, 8);
24     printf("Push success=%d, top=%d\n", flag, top);
25     flag = pop(stack, &top, &data);
26     printf("Pop success=%d, data=%d, top=%d\n", flag, data, top);
27     flag = pop(stack, &top, &data);
28     printf("Pop success=%d, data=%d, top=%d\n", flag, data, top);
29     flag = pop(stack, &top, &data);
30     printf("Pop success=%d, data=%d, top=%d\n", flag, data, top);
31     return 0;
32 }
```

3. Fill in the space to complete the code. (9%)

(1) ____(*talk)_____ (2) ___talkRich___ (3) ___talkHungry___

output	Luna 1000 Mimi 5

```
01 #include <string.h>
02 typedef struct cat {
03     char name[10];
04     void _____(1)_____ (struct cat*, int money);
05 } cat_t;
06 void talkRich(cat_t* c, int money) {
07     if (money > 100) {
08         printf("%s %d\n", c->name, money);
09     } else {
10         printf("Meow\n");
11     }
12 }
13 void talkHungry(cat_t* c, int money) {
14     if (money > 10) {
15         printf("Meow\n");
16     } else {
17         printf("%s %d\n", c->name, money);
18     }
19 }
20 void talkMeow(cat_t* c, int money) {
21     printf("%s Meow %d\n", c->name, money);
22 }
23 int main() {
24     cat_t cat1, cat2;
25     strcpy(cat1.name, "Luna");
26     cat1.talk = _____(2)_____;
27     strcpy(cat2.name, "Mimi");
28     cat2.talk = _____(3)_____ ;
29     cat1.talk(&cat1, 1000);
30     cat2.talk(&cat2, 5);
31     return 0;
32 }
```

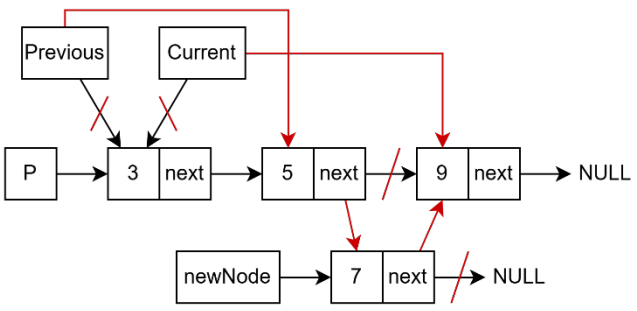
4. Fill in the space to complete the code. **(9%)**

(1) _tail+1/++tail_ (2) _head_ (3) __index[1]+1/++index[1]__

Output	enqueue 0 enqueue 1 enqueue 2 queue full queue full

```
01 #define SIZE 4
02 typedef enum { FALSE, TRUE } bool;
03 bool isFull(int head, int tail) {
04     return (((_____(1)_____) % SIZE) == ____(2)____);
05 }
06 bool enqueue(int data[], int index[], int key) {
07     if (isFull(index[0], index[1])) return FALSE;
08     index[1] = (_____(3)_____) % SIZE;
09     data[index[1]] = key;
10     return TRUE;
11 }
12 int main() {
13     int k = 0, i = 0, index[3] = {0, 0, 0}, data[SIZE];
14     bool result;
15     for (i = 0; i < 5; i++) {
16         result = enqueue(data, index, k++);
17         if (!result)
18             printf("queue full\n");
19         else
20             printf("enqueue %d\n", k - 1);
21     }
22 }
```

5. To insert the "**newNode 7**" into the linked list **in order.** Please complete the following figure and **show the modification of each pointer**. **(5%)**



6. Please describe your learning problem and how to improve it. (30 or more word will be scored) **(5%)**

7. Fill the best one of the following: (A) abstraction (B) coupling (C) implementation, (D) interface, (E) refactoring, (F) scalability, (G) encapsulation, (H) maintainability, (I) flexibility, (J) structure. **(10%)**

(1) Abstract Data Types define the __D__ of operations that can be performed on a piece of data, without revealing the internal __C__.

(2) __G__ means the internal representation of the data and the implementation of operations are hidden from the user.

(3) High __H__ means a program is easy to repair, add, and modify.

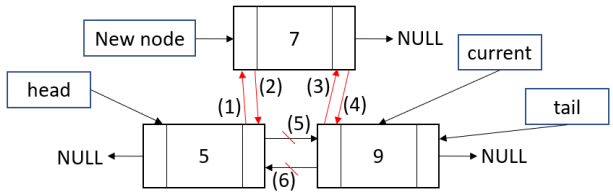
(4) High __I__ means the ease with which code can be adapted and extended without significantly impacting existing functionality or requiring extensive changes, and allowing for quicker adaptation to changing requirements.

(5) __E__ is the process of restructuring existing code to improve its design, structure, and/or implementation without changing its external behavior. It aims to enhance code quality, and readability while preserving functionality.

(6) High __B__ code means function or components in a program are heavily dependent on each other.

8. Please answer which line of the code corresponds to each operation when inserting the value 7 into the double linked list. **(12%)**

(1)__32__ (2)__31__ (3)__33__ (4)__30__ (5)__32__ (6)__33__




```
01 #include <stdlib.h>
02 typedef struct dnode_s {
03       int data;
04       struct dnode_s *front;
05       struct dnode_s *back;
06 } node_t;
07 typedef node_t *nodep_t;
08 void insert_sorted(nodep_t *headp, nodep_t *tailp, int value) {
09    nodep_t new_node = (nodep_t)malloc(sizeof(node_t));
10    new_node->data = value;
11    new_node->front = NULL;
12    new_node->back = NULL;
13    if (*headp == NULL) {
14       *headp = *tailp = new_node;
15       return;
16    }
17    nodep_t current = *headp;
18    while (current && current->data < value) {
19       current = current->front;
20    }
21    if (current == NULL) {
22       (*tailp)->front = new_node;
23       new_node->back = *tailp;
24       *tailp = new_node;
25    } else if (current->back == NULL) {
26       new_node->front = current;
27       current->back = new_node;
28       *headp = new_node;
29    } else {
30       new_node->front = current;
31       new_node->back = current->back;
32       current->back->front = new_node;
33       current->back = new_node;
34    }
35 }
36 int main() {
37    nodep_t head = NULL, tail = NULL;
38    insert_sorted(&head, &tail, 5);
39    insert_sorted(&head, &tail, 9);
40    insert_sorted(&head, &tail, 7);
41    return 0;
42 }
```

9. Fill in the space to complete the code. **(9%)**

(1) ____< / <=____ (2) ___p->next___ (3) _____p_____

Output	10 → 30 → 50 10 → 30 → 40 → 50 5 → 10 → 30 → 40 → 50

```
01 #include <stdlib.h>
02 typedef struct node {
03       int data;
04       struct node* next;
05 } node_t, *nodep_t;
06 nodep_t create(int data) {
07    nodep_t newp = (nodep_t)malloc(sizeof(node_t));
08    newp->data = data;
09    newp->next = NULL;
10    return newp;
11 }
12 void printList(nodep_t p) {
13    while (p != NULL) {
14       printf("%d", p->data);
15       if (p->next != NULL) printf(" → ");
16       p = p->next;
17    }   printf("\n");
18 }
19 nodep_t insertInOrderR(nodep_t p, int data) {
20    nodep_t newp;
21    if (p == NULL || data ____(1)____ p->data) {
22       newp = create(data);
23       newp->next = p;
24       return newp;
25    } else {
26       p->next = insertInOrderR(____(2)____, data);
27       return ____(3)____;
28    }
29 }
30 int main() {
31    nodep_t head = NULL;
32    head = insertInOrderR(head, 10);
33    head = insertInOrderR(head, 30);
34    head = insertInOrderR(head, 50);
35    printList(head);
36    head = insertInOrderR(head, 40);
37    printList(head);
38    head = insertInOrderR(head, 5);
39    printList(head);
40    return 0;  }
```

10. Fill in the space to complete the code. **(9%)**

(1) _____fopen_____ (2) ____fputc_____ (3) _____fclose_____

```
01 #include <stdlib.h>
02 int main() {
03    FILE *inFile, *outFile;
04    char ch;
05    inFile = ___(1)___("input.txt", "r"); // Open input.txt for reading
06    if (inFile == NULL) {
07       printf("Cannot open input file\n");
08       return 1;
09    }
10    outFile = ___(1)___("output.txt", "w"); // Open output.txt for writing
11    while ((ch = fgetc(inFile)) != EOF) {
12       if (ch == 'a') ch = '@';
13       ___(2)___(ch, outFile); // Write character ch to output.txt
14    }
15    ___(3)___(inFile);  // Close input.txt
16    ___(3)___(outFile); // Close output.txt
17    return 0; }
```

11. Briefly describe the meaning of the code. (6%)

```
01 #include <string.h>
02 typedef struct student_s {   char name[8];    int id; } student_t;
03 void fun(char fileName[]) {
04    FILE *fp;
05    student_t s2, s1 ={"John", 116590011};
06    fp = fopen(fileName, "wb+");
07    fwrite(&s1, sizeof(student_t), 1, fp);
08    fseek(fp, 0, SEEK_END);              // (1)
09    long lSize = ftell(fp);              // (2)
10    rewind(fp);                          // (3)
11    fread(&s2, sizeof(student_t), 1, fp);
12    printf("%s, %d, %.2f\n", s2.name, s2.id, s2.score);
13    fclose(fp);
14 }
15 int main() {   fun("a.bin");   return(0);}
```

(1) ___檔案指標指到檔案結尾_____

(2) ___目前檔案指標位置距離_____

(3) ____檔案指標回到檔案開頭_____