2022 I2P(I) Written Exam

12/26 (Mon.) 15:30-17:30

True and False (15%)

Write `T` if the description is correct, `F` otherwise. No explanation is required. (`+1` for each correct answer)

1. To check whether $a \in (2, 8)$, we can write following code in C language: `if (2 < a < 8)`.
2. When reading `EOF`, `scanf` and `getchar` returns `EOF`, while `gets` and `fgets` return `NULL`.
3. The return value of `scanf` indicates the number of variables that have been read successfully.
4. `do-while` works exactly the same as `while`, that is why we rarely use them in our code.
5. `if ((a != 0) & (2 / a == 1))` won't result in divide-by-zero error even if `a` is zero due to the shortcut effect of `&`.
6. An infinity loop `while(1);` is dangerous and will make the program crash within a few minutes.
7. Misusing pointers can pose a serious security threat to programs. You'll often see Buffer Overflow security fixes in Google Chrome, Mozilla Firefox, or any other applications. These security issue are mostly caused by pointers. Thus, in many high-level languages, the usage of pointer is forbidden.
8. We can use `getchar` and some post-process code to replace any `scanf` related input.
9. `while(--T) {...}` can be used to repeat a piece of code for exactly $T - 1$ times.
10. If we want to write recursion codes like writing mathematical induction proofs, we only need to check the recursive cases are true, no matter the basic case is true or not.
11. Any `while`-loop can be replaced by recursion code, and any recursion code can be replaced by `while`-loop.
12. The following code is incorrect, and if we replace `str` with `&str` can solve the problem.

    ```c
    char str[10];
    scanf("%s", str);
    ```

13. Assume that `sizeof(int)` is 4 and `sizeof(int*)` is 8. The result of the code below is `4` since the pointer increases by `sizeof(int)`.

    ```c
    int n;
    int* ptr = &n;
    printf("%d\n", (int)(ptr + 1) - (int)ptr);
    ```

14. Arrays are pointers.
15. `sizeof(float*)` and `sizeof(int***)` have the same size since pointers are just variables that store memory addresses.

Binary Conversion (30%)

Answer the following questions. (`+3` for each correct answer.)

1. Convert 111_{10} to its equivalent binary form with a total of 8-bits.
2. Convert 11011110_2 to its equivalent base ten form.
3. Convert -26_{10} to its equivalent two's complement form with a total of 8-bits.
4. Compute the negative value of 01101111_2 in two's complement form with a total of 8-bits.
5. In binary representation, what is 010100110110_2 + 010010101011_2 ?
6. What is $0x98$ + $0x59$ in hexadecimal form?
7. What is the hexadecimal representation of 0100111011111101_2?
8. If we have 4-bits as fraction bits and 3 bits as exponent bits, what is the binary representation of the floating-point 2.271 using IEEE standard for floating-point? (Direct truncation, no rounding)
9. If we have 4-bits as fraction bits and 3 bits as exponent bits, what is the binary representation of the floating-point -0.832 using IEEE standard for floating-point? (Direct truncation, no rounding)
10. Following the previous question, what is the value of truncation error when encoding 3.28 using IEEE standard for floating-point?

The exponents of question 8-10 use Excess-4 notation, as shown in the table below:

Bit Pattern	Excess-4 Notation
111	3
110	2
101	1
100	0
011	-1
010	-2
001	-3
000	-4

The binary representation is as follows:

Sign	Exponent	Fraction
1-bit	3-bit	4-bit

Compute by Hand (25%)

If the program ran without error, provide the output for each question. If the program includes undefined behaviors or errors, point out the first encountered issue when running the program (e.g., in line `x`, during the `i`-th iteration of the loop) and explain why the issue occurs. (`+5` for the correct answer.)

1. code

```c
#include <stdio.h>

int main(void) {
  char a, b;
  int ret;
  a = b = -1;
  ret = scanf("(%c %c)", &a, &b);
  printf("%d %d %d", ret, a, b);
  return 0;
}
```

input

```
(A
C)
```

Hint: 'A' is 65, '\n' is 10.

2. code

```c
#include <stdio.h>

int main(){
    char s[6] = "glean";
    int T = 3, a;
    while(--T){
        scanf("%d", &a);
        scanf("%c", s+a);
    }

    printf("%s", s);
}
```

input

```
1r3e4d
```

3. code

```c
#include <stdio.h>

int main(void) {
    int arr[5] = {5, 4, 3, 2, 1};
    int a = 0;
    int b = 0;

    switch(*(arr+2)){
        case 1:
            printf("%d\n", 1<<2);
        case 2:
            a = b ? 5 : 6;
            printf("%d\n", a);
        case 3:
            b = (a=3<5) ? 1 : -1;
            printf("%d\n", a);
        case 4:
            printf("%d\n", b);
        case 5:
            printf("%d\n", 5<<2);
        default:
            puts("Default");
    }
    putchar('!');
    return 0;
}
```

4. code

```c
#include <stdio.h>
#define MAGIC(X,Y,Z) X-Y+Z

int main(void) {
    int ans = 8 / MAGIC(4, 3, 1) * 5;
    printf("%03d", ans);
    return 0;
}
```

5. code

```c
#include <stdio.h>
#include <string.h>
#define swap(a, b) if (a != b) {(*a)^=(*b); (*b)^=(*a); (*a)^=(*b);}

void banana_swap(char* head, char* end, int level){
    if(level > 2) return;
    char *a = head, *b = end;

    while(*a < *end) a++;
    while(*b > *head) b--;

    swap(a, b);
    if(level == 2) a = b;

    printf("level%d: %s\n", level, head);

    banana_swap(head, end, level+1);
}

int main(){
    char s[10] = "mean";
    banana_swap(s, s+(strlen(s)-1), 1);
    printf("level0: %s", s);
}
```

Debug Practice (30%)

Each code snippet below has an unknown number of bugs. Try insert/remove the least number of characters to fix all bugs in the code.

For each problem, write down the line with issues and correct it. You should also mark the line with issues on the exam paper, since TAs will refer to your exam paper if your answer is ambiguous. The bugs may include compile errors, runtime undefined behaviors, memory leaking issues, etc.

1. Second Highest Value (5%)

   Given an array with length of N.
   $\forall i \in [1, N]$, you have to output the second highest value in the subarray $[1, i]$ (i.e. the second integer after sorting the subarray from higher to lower)

   If there are less than two numbers in the subarray, you should print 0 instead.

   Example: N = 3, array = 5 7 9

   - when i = 1, there are less than two numbers in the subarray $[1, 1]$(i.e. subarray = 5), so print 0.
   - when i = 2, 5 is the second highest value in the subarray $[1, 2]$(i.e. subarray = 5 7), so print 5.
   - when i = 3, 7 is the second highest value in the subarray $[1, 3]$(i.e. subarray = 5 7 9), so print 7.

   ⇒ output is 0 5 7

```c
#include <stdio.h>

int main()
{
    int N, input;
    int biggest = second = 0;
    scanf("%d", &N);
    for(int i = 0; i < N; ++i) {
        scanf("%d", &input);
        if(input >= biggest) {
            biggest = input;
        }
        else if(input > second) {
            second = input;
        }
        if(i != N-1)
            printf("%d ", second);
        else
            printf("%d\n", second);
    }

    return 0;
}
```

   Sample Input:

```
5
10 7 8 3 2
```

Sample Output:

```
0 7 8 8 8
```

2. Split string (5%)

```c
#include <stdio.h>

int main(){
    char A[100];
    scanf("%s", A);

    int i;
    for (i = 0; A[i] != ' '; ++i)
        printf("%c", A[i]);
    puts("");

    for (i ; A[i] != '\0'; ++i)
        printf("%c", *(A+i));

    return 0;
}
```

Sample Input:

```
abc 123
```

Sample Output:

```
abc
123
```

3. Dynamic 3D array (5%)

   In this problem, you are asked to create an n * m * k 3D unsigned array and free the memory space correctly.

   Note that you don't have to check the main function and the Random function.

```
#include<stdio.h>
#include<stdlib.h>

unsigned random_seed=7122;
unsigned Random(){
    return random_seed=random_seed*0xdefaced+1;
}
unsigned*** new_3d_array(unsigned n,unsigned m,unsigned k)
{
    unsigned* lk;
    unsigned** lm;
    unsigned*** ln;

    unsigned i,j;

    lk = (unsigned *) malloc(sizeof(unsigned)*n*m*k);
    lm = (unsigned **) malloc(sizeof(unsigned*)*n*m);
    ln = (unsigned ***) malloc(sizeof(unsigned**)*n);

    for(i=0;i<n;i++)
    {
        ln[i]=lm+i*m;
        for(j=0;j<m;j++)
        {
            lm[i*m+j]=lk+i*m+j*k;
        }
    }
    return ln;
}
void delete_3d_array(unsigned ***arr)
{
    free(arr);
    free(arr[0]);
    free(arr[0][0]);
}

int main()
{
    int n,m,k;
    scanf("%d%d%d",&n,&m,&k);
    unsigned ***arr=new_3d_array(n,m,k);
    int i,j,l;
    for(i=0;i<n;++i)
    {
        for(j=0;j<m;++j)
            for(l=0;l<k;++l)
                arr[i][j][l]=Random();
    }
    delete_3d_array(arr);
```

```
    return 0;
}
```

Sample Input:

```
100 100 100
```

Sample Output:

```

```

4. Full permutation (5%)

   Given a sorted string **s**, find out all the different permutations of **s**. (including **s** itself)

```c
#include <stdio.h>
#include <string.h>

char s[11];
char res[11];
int visited[11];

void perm();

int main(void)
{
    scanf("%s", s);
    perm(0,strlen(s));
}

void perm(int pos, int len)
{
    int i;
    //basis step
    if(pos==len)
    {
        res[pos]='\0';
        printf("%s\n",res);

        return;
    }

    char head='\0';
    //recursive step
    for(i=0;i<len;i++)
    {
        if(!visited[i])
        {
            if(s[i]!=head) head=s[i];
            else break;

            res[pos]=s[i];
            perm(pos+1,len);
        }
    }
}
```

   Sample Input:

```
aabc
```

   Sample Output:

```
aabc
aacb
abac
abca
acab
acba
baac
baca
bcaa
caab
caba
cbaa
```

5. String sorting (10%)

   Given several strings, please output them in **alphabetical order** (one string per line).

   It is guaranteed that

   - # of strings won't exceed 200000
   - Length of each string won't exceed 100

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define COLS 101

char input[200000][COLS];
int len;

void bubble_sort(char** ap, int rows) {
    int i, j;
    char temp[COLS];

    for (i = 0; i < rows-1; i++) {
        for (j = 0; j < rows-i; j++) {
            if (strcmp(ap[j], ap[j+1]) < 0) {
                temp = ap[j];
                ap[j] = ap[j+1];
                ap[j+1] = temp;
            }
        }
    }
}

int main(void) {
    int i;

    while(scanf("%s", input[len++])!=EOF);
    bubble_sort(input, len);
    for(i=0; i < len; i++) printf("%s\n", input[i]);

    return 0;
}
```

Sample Input:

```
acbd
hgfe
abcd
zgfe
```

Sample Output:

```
abcd
acbd
hgfe
zgfe
```