考試用函數庫

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<time.h>
#include<stddef.h>//怕出錯就加上
```

數組

數組要擔心的就是内存爆炸。最好每個數組最後一位都要設置

```
char chara[8];
//一堆輸入
chara[8] = '\0';
```

這樣在把chara當成string輸出的時候才不會輸出奇怪的東西。

數據類型

一直很擔心什麽會報錯什麽不會報錯，運算結果有哪些不對的。

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int fdevidei = 3.3/3;
    printf("%d\n", fdevidei);//1
    printf("%f\n", fdevidei);//0

    float f_devidei = 2.1/2;
    printf("%d\n", f_devidei);//The address
    printf("%f\n", f_devidei);//1.05

    int idevf = 1/0.5;
    printf("%d\n", idevf);//2
    printf("%f\n", idevf);//0

    float i_devidef = 2.3/2;
    printf("%d\n", i_devidef);//The address
    printf("%f\n", i_devidef);//1.15
 return 0;
}
```

注意：很簡單的一個東西。只要定義的和打印的數據類型一致就不會報錯。int是只保留整數位，float保留小數。

int	float	char	string	double
%d	%f （%.nf顯示n位數字）	%c	%s	%l(e,f,g)可以通過e±將科學計數法變成數字。%lf (%.nlf顯示n位數字)

C語言中沒有string。創建方法可以是開闢一個無限大的char型數組。若在global中定義并輸入的時候不需要規定其大小，但是在函數中定義式需要規定大小。

getchar()，getch()和scanf("%c", char*)的區別

getch()不同編譯器不同。一些編譯器需`#include <conio.h>`
scanf是讀到空格或者\n\0截止。而getch()和getchar()都會讀取空格或者換行。不同的是getch()是讀取后直接打印，getchar()返回重新輸入的第一個字符。
getchar()可以用來清內存不報錯。用

```
while((str2 = getchar() )!= '\n' && str2 != EOF)
```

可以清除緩存區的內存。

條件判斷

switch case, if, else if, while, do while, for

```
switch(condition){
    case(val1):
    {Action1;
    break;}
}

if(condition1){
    Action;
}else if(condition2){
    Action;
}else{Action;}
\\if和else if是不能用break和continue的

while(condition){
    Action;
}

do{
    Action;
}while(condition);
\\do while會先執行do裏面的内容再判斷是否到條件。到了就不do了。
\\沒到返回do。
\\注意：do之後的數據不會先存儲進去。會先計算但不存儲。

for(variable; condition; action){
    Actions;
}
```

指針(以下列舉的都是出數字的部分)

int a = 5;
int *b;
b = &a;
printf("%d", *b);

```
int c[3] = {1,2,3};
int *cPtr;
cPtr = c;
//此時對cPtr進行計算，只是對其地址進行運算。
//真正的數值運算還是需要cPtr[]進行計算
```

FILE(個人覺得最難)

先說這個wra和w+,r+和a+。網上一些blog上面說+的功能更强大。但是win11 codeblocks上體驗到的功能是一樣的，注意事項也是一樣的。

w	r	a
新建/overwrite文件	只讀，文件必須存在	在文件後續寫/新建文件

1. fopen(const char *filename, const char *mode)

   ```
   FILE *filePointer;
   filePointer = fopen( “existing file.txt”, “r”);
   ```

2. fclose()

   ```
   fclose(filePointer);
   ```

3. fprintf()

   ```
   fprintf( filePointer, “%d\n”, 98 );
   ```

4. fscanf()

   ```
   int number;
   fscanf( filePointer, “%d”, &number );
   ```

5. feof()用於判斷是否輸入完全，特別好用高級

```
feof( filePointer );

//for example
while(!feof(filePointer)){
   Action;
}
//win Ctrl+z退出，Mac Ctrl+d退出
```

特别奇怪。不知道爲什麽不可以寫成feof(filePointer) != NULL，而必須寫成!feof(filePointer) 。

6. fgets()

```
char myString[50];
fgets( myString, 49, filePointer );
```

7. fputs()

```
FILE *dPtr;
char str1[50];
dPtr = fopen("test.txt", "r");
fgets(str1, 49, dPtr);
printf("%s", str1);
```

8. fgetc()

```
fgetc(filePointer);
```

9. fputc()

```
char str = 'c';
fPtr = fopen("test.txt", "w");
fputc(str, filePointer);//等同於fputc('c', filePointer);
```

特别注意！！！scan和get不要同時用。要用就一用到底。同樣，print和put不要同時使用，要用也要一用到底。否則容易出現打印錯誤。

string的一些函數（個人認爲第二大難點）

需要`#include <string.h>`

1. sprintf()

   char str[80];
   sprintf(str, "Hello");
   //也等同於
   //sprintf(str, "%s", "Hello");

2. sscanf()

   sscanf(s, "%d", &x);

3. strcpy()

char str[80];
sprintf(str, "Hello");
char str1[80];
strcpy(str1, str);
puts(str1);//顯示Hello在終端

4. strncpy()

   char str[80];
   sprintf(str, "Hello F");
   char str1[80];
   strncpy(str1, str, 7);
   puts(str1);//顯示Hello F在終端
   會讀取空格。後面數字代表讀取幾個字符。

5. strlen()

   strlen(str);
   顯示的數據類型是unsigned int

6. strcat()

```
strcat(char *dest, char *src);
```

還有各類ncmp， ncpy， ncat都不詳細介紹了。主要就是前n位數作比較的事情。

Structure

一個非常好的例子。

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


struct student
{
    int id;
    char name[30];
    float percentage;
};

int main()
{
    int i;
    struct student record1 = {1, "Marry", 70.5};
    struct student *record1ptr;
    struct student record2, record4;

    printf("Records of STUDENT1 - record1 structure \n");
    printf("  Id : %d \n  Name : %s\n  Percentage : %f\n",
            record1.id, record1.name, record1.percentage);

    record1ptr = &record1;
```

```
    printf("Records of STUDENT1 - record1 structure pointer \n");
    printf("  Id : %d \n  Name : %s\n  Percentage : %f\n",
            record1ptr->id, record1ptr->name, record1ptr->percentage);

    // 1st method to copy whole structure to another structure
    record2=record1;

    printf("\nRecords of STUDENT1 - Direct copy from " \
           "record1 \n");
    printf("  Id : %d \n  Name : %s\n  Percentage : %f\n",
            record2.id, record2.name, record2.percentage);

    // 2nd method to copy by individual members
    printf("\nRecords of STUDENT1 - Copied individual " \
           "members from record1 \n");
    record4.id=record1.id;
    strcpy(record4.name, record1.name);
    record4.percentage = record1.percentage;

    printf("  Id : %d \n  Name : %s\n  Percentage : %f\n",
            record4.id, record4.name, record4.percentage);

    return 0;
}
```

當然，日常使用的時候，由於本人懶狗，直接會

```
typedef struct{
    int id;
    char name[80];
    float percentage;
} Student;//一個有八十個學生的struct
```

第一個例子對於要複製一整個struct來說比較方便，第二個例子對於不同的輸入會比較方便。

```
#include <stdio.h>

// 使用 typedef 为结构体命名
typedef struct {
    int id;
    char name[80];
    float percentage;
} Student; // 使用 Student 替代 struct students

int main() {
    // 使用 typedef 后的结构体声明数组
    Student toomany[80]; // 一个有80个学生的struct数组

    printf("Enter information for 80 students:\n");

    for (int i = 0; i < 80; i++) {
```

```
        printf("Enter details for student %d:\n", i + 1);

        printf("ID: ");
        scanf("%d", &toomany[i].id);

        printf("Name: ");
        scanf("%s", toomany[i].name);

        printf("Percentage: ");
        scanf("%f", &toomany[i].percentage);
    }

    printf("\nStudent Details:\n");

    for (int i = 0; i < 80; i++) {
        printf("ID: %d, Name: %s, Percentage: %.2f\n", toomany[i].id,
toomany[i].name, toomany[i].percentage);
    }

    return 0;
}
```

在第一個例子的Ptr中，Ptr部分的知識和前文提到的Ptr是一樣的。不用擔心太多。

TimeFunction（碎碎念：真的感覺是老師CW1沒講然後乾脆上一個補償措施）

包含于`time.h`庫中。可調取系統時間。

1. `time_t time (time_t* timer);`

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

int main() {
    // 声明一个 time_t 类型的变量用于存储时间
    time_t currentTime;

    // 获取当前时间，并将其存储在 currentTime 变量中
    time(&currentTime);

    // 输出当前时间的秒数。從1970.1.1開始的
    printf("Current time: %ld\n", currentTime);

    // 可以使用 ctime 函数将时间转换为字符串并输出
    printf("Current time as string: %s", ctime(&currentTime));

    return 0;
}
```

2. `struct tm * localtime (const time_t * timer);`

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

int main() {
    // 声明一个 time_t 类型的变量用于存储时间
    time_t currentTime;

    // 获取当前时间，并将其存储在 currentTime 变量中
    time(&currentTime);

    // 将时间转换为本地时间的 struct tm 结构体
    struct tm *localTime = localtime(&currentTime);

    // 输出本地时间的各个元素
    printf("Year: %d\n", localTime->tm_year + 1900);  // tm_year表示的是从1900年开始的
年数
    printf("Month: %d\n", localTime->tm_mon + 1);     // tm_mon表示的是月份，范围是0-11
    printf("Day: %d\n", localTime->tm_mday);
    printf("Hour: %d\n", localTime->tm_hour);
    printf("Minute: %d\n", localTime->tm_min);
    printf("Second: %d\n", localTime->tm_sec);

    return 0;
}
```

3. `time_t mktime (struct tm * timeptr);`

上面的指針是從系統讀取時間。這個用於自己設定時間然後轉化。

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

int main() {
    // 声明一个 struct tm 结构体变量用于表示时间
    struct tm myTime;

    // 设置 struct tm 结构体的各个元素
    myTime.tm_year = 2023 - 1900; // 年份需减去 1900
    myTime.tm_mon = 0;            // 月份范围是 0-11，0 表示 1 月
    myTime.tm_mday = 1;           // 日期
    myTime.tm_hour = 12;          // 小时
    myTime.tm_min = 0;            // 分钟
    myTime.tm_sec = 0;            // 秒数
    myTime.tm_isdst = -1;         // 是否为夏令时，-1 表示未知

    // 使用 mktime 将 struct tm 结构体转换为 time_t 类型的时间
    time_t convertedTime = mktime(&myTime);

    // 输出转换后的时间
    printf("Converted time: %ld\n", convertedTime);
```

```
    return 0;
}
```

4. `char* asctime (const struct tm * timeptr);`

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

int main() {
    // 获取当前时间
    time_t currentTime;
    time(&currentTime);

    // 将当前时间转换为本地时间的 struct tm 结构体
    struct tm *localTime = localtime(&currentTime);

    // 检查 struct tm 结构体是否为空
    if (localTime != NULL) {
        // 使用 asctime 将 struct tm 结构体转换为字符串
        char *timeString = asctime(localTime);

        // 检查转换后的时间字符串是否为空
        if (timeString != NULL) {
            // 输出转换后的时间字符串
            printf("Converted time string: %s", timeString);
        } else {
            // 打印错误信息
            perror("asctime");
        }
    } else {
        // 打印错误信息
        perror("localtime");
    }

    return 0;
}
```

Rand()

按照等差數列生成數。

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


int main()
{
   int i, n;
```

```
   n = 5;

   /* 初始化随机数发生器 */
   srand((unsigned) time(NULL));

   /* 输出 13 到 63 之间的 5 个随机数 */
   for( i = 0 ; i < n ; i++ ) {
      printf("%d\n", rand() % 51+13);
   }

  return(0);
}
```

十大經典排序方式

1. 冒泡排序

```
//從小到大
#include <stdio.h>
void bubble_sort(int arr[], int len) {
        int i, j, temp;
        for (i = 0; i < len - 1; i++)
                for (j = 0; j < len - 1 - i; j++)
                        if (arr[j] > arr[j + 1]) {
                                temp = arr[j];
                                arr[j] = arr[j + 1];
                                arr[j + 1] = temp;
                        }
}
```

2. 選擇排序

   首先在未排序序列中找到最小（大）元素，存放到排序序列的起始位置。

   再从剩余未排序元素中继续寻找最小（大）元素，然后放到已排序序列的末尾。

   重复第二步，直到所有元素均排序完毕。

```
void swap(int *a,int *b) //交換兩個變數
{
    int temp = *a;
    *a = *b;
    *b = temp;
}
void selection_sort(int arr[], int len)
{
    int i,j;

        for (i = 0 ; i < len - 1 ; i++)
    {
```

```
                int min = i;
                for (j = i + 1; j < len; j++)     //走訪未排序的元素
                        if (arr[j] < arr[min])    //找到目前最小值
                                min = j;    //紀錄最小值
                swap(&arr[min], &arr[i]);    //做交換
        }
}
```

3. 插入排序

将第一待排序序列第一个元素看做一个有序序列，把第二个元素到最后一个元素当成是未排序序列。

从头到尾依次扫描未排序序列，将扫描到的每个元素插入有序序列的适当位置。（如果待插入的元素与有序序列中的某个元素相等，则将待插入元素插入到相等元素的后面。）

```
void insertion_sort(int arr[], int len){
        int i,j,key;
        for (i=1;i<len;i++){
                key = arr[i];
                j=i-1;
                while((j>=0) && (arr[j]>key)) {
                        arr[j+1] = arr[j];
                        j--;
                }
                arr[j+1] = key;
        }
}
```

4. 希爾排序

```
void shell_sort(int arr[], int len) {
        int gap, i, j;
        int temp;
        for (gap = len >> 1; gap > 0; gap >>= 1)
                for (i = gap; i < len; i++) {
                        temp = arr[i];
                        for (j = i - gap; j >= 0 && arr[j] > temp; j -= gap)
                                arr[j + gap] = arr[j];
                        arr[j + gap] = temp;
                }
}
```

5. 歸并排序

```
int min(int x, int y) {
    return x < y ? x : y;
}
void merge_sort(int arr[], int len) {
    int *a = arr;
    int *b = (int *) malloc(len * sizeof(int));
    int seg, start;
```

```
    for (seg = 1; seg < len; seg += seg) {
        for (start = 0; start < len; start += seg * 2) {
            int low = start, mid = min(start + seg, len), high = min(start + seg *
2, len);
            int k = low;
            int start1 = low, end1 = mid;
            int start2 = mid, end2 = high;
            while (start1 < end1 && start2 < end2)
                b[k++] = a[start1] < a[start2] ? a[start1++] : a[start2++];
            while (start1 < end1)
                b[k++] = a[start1++];
            while (start2 < end2)
                b[k++] = a[start2++];
        }
        int *temp = a;
        a = b;
        b = temp;
    }
    if (a != arr) {
        int i;
        for (i = 0; i < len; i++)
            b[i] = a[i];
        b = a;
    }
    free(b);
}
```

6. 快速排序

```
typedef struct _Range {
    int start, end;
} Range;

Range new_Range(int s, int e) {
    Range r;
    r.start = s;
    r.end = e;
    return r;
}

void swap(int *x, int *y) {
    int t = *x;
    *x = *y;
    *y = t;
}

void quick_sort(int arr[], const int len) {
    if (len <= 0)
        return; // 避免len等於負值時引發段錯誤（Segment Fault）
    // r[]模擬列表,p為數量,r[p++]為push,r[--p]為pop且取得元素
    Range r[len];
```

```
    int p = 0;
    r[p++] = new_Range(0, len - 1);
    while (p) {
        Range range = r[--p];
        if (range.start >= range.end)
            continue;
        int mid = arr[(range.start + range.end) / 2]; // 選取中間點為基準點
        int left = range.start, right = range.end;
        do {
            while (arr[left] < mid) ++left;   // 檢測基準點左側是否符合要求
            while (arr[right] > mid) --right; //檢測基準點右側是否符合要求
            if (left <= right) {
                swap(&arr[left], &arr[right]);
                left++;
                right--;               // 移動指針以繼續
            }
        } while (left <= right);
        if (range.start < right) r[p++] = new_Range(range.start, right);
        if (range.end > left) r[p++] = new_Range(left, range.end);
    }
}
```

7. 堆排序

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

void swap(int *a, int *b) {
    int temp = *b;
    *b = *a;
    *a = temp;
}

void max_heapify(int arr[], int start, int end) {
    // 建立父節點指標和子節點指標
    int dad = start;
    int son = dad * 2 + 1;
    while (son <= end) { // 若子節點指標在範圍內才做比較
        if (son + 1 <= end && arr[son] < arr[son + 1]) // 先比較兩個子節點大小，選擇最大
的
            son++;
        if (arr[dad] > arr[son]) //如果父節點大於子節點代表調整完畢，直接跳出函數
            return;
        else { // 否則交換父子內容再繼續子節點和孫節點比較
            swap(&arr[dad], &arr[son]);
            dad = son;
            son = dad * 2 + 1;
        }
    }
}
```

```c
void heap_sort(int arr[], int len) {
    int i;
    // 初始化，i從最後一個父節點開始調整
    for (i = len / 2 - 1; i >= 0; i--)
        max_heapify(arr, i, len - 1);
    // 先將第一個元素和已排好元素前一位做交換，再重新調整，直到排序完畢
    for (i = len - 1; i > 0; i--) {
        swap(&arr[0], &arr[i]);
        max_heapify(arr, 0, i - 1);
    }
}

int main() {
    int arr[] = { 3, 5, 3, 0, 8, 6, 1, 5, 8, 6, 2, 4, 9, 4, 7, 0, 1, 8, 9, 7, 3, 1,
2, 5, 9, 7, 4, 0, 2, 6 };
    int len = (int) sizeof(arr) / sizeof(*arr);
    heap_sort(arr, len);
    int i;
    for (i = 0; i < len; i++)
        printf("%d ", arr[i]);
    printf("\n");
    return 0;
}
```

8. 計數排序

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

void print_arr(int *arr, int n) {
        int i;
        printf("%d", arr[0]);
        for (i = 1; i < n; i++)
                printf(" %d", arr[i]);
        printf("\n");
}

void counting_sort(int *ini_arr, int *sorted_arr, int n) {
        int *count_arr = (int *) malloc(sizeof(int) * 100);
        int i, j, k;
        for (k = 0; k < 100; k++)
                count_arr[k] = 0;
        for (i = 0; i < n; i++)
                count_arr[ini_arr[i]]++;
        for (k = 1; k < 100; k++)
                count_arr[k] += count_arr[k - 1];
        for (j = n; j > 0; j--)
                sorted_arr[--count_arr[ini_arr[j - 1]]] = ini_arr[j - 1];
        free(count_arr);
}
```

```
int main(int argc, char **argv) {
        int n = 10;
        int i;
        int *arr = (int *) malloc(sizeof(int) * n);
        int *sorted_arr = (int *) malloc(sizeof(int) * n);
        srand(time(0));
        for (i = 0; i < n; i++)
                arr[i] = rand() % 100;
        printf("ini_array: ");
        print_arr(arr, n);
        counting_sort(arr, sorted_arr, n);
        printf("sorted_array: ");
        print_arr(sorted_arr, n);
        free(arr);
        free(sorted_arr);
        return 0;
}
```

9. 桶排序

```
#include<iterator>
#include<iostream>
#include<vector>
using namespace std;
const int BUCKET_NUM = 10;

struct ListNode{
        explicit ListNode(int i=0):mData(i),mNext(NULL){}
        ListNode* mNext;
        int mData;
};

ListNode* insert(ListNode* head,int val){
        ListNode dummyNode;
        ListNode *newNode = new ListNode(val);
        ListNode *pre,*curr;
        dummyNode.mNext = head;
        pre = &dummyNode;
        curr = head;
        while(NULL!=curr && curr->mData<=val){
                pre = curr;
                curr = curr->mNext;
        }
        newNode->mNext = curr;
        pre->mNext = newNode;
        return dummyNode.mNext;
}


ListNode* Merge(ListNode *head1,ListNode *head2){
```

```
        ListNode dummyNode;
        ListNode *dummy = &dummyNode;
        while(NULL!=head1 && NULL!=head2){
                if(head1->mData <= head2->mData){
                        dummy->mNext = head1;
                        head1 = head1->mNext;
                }else{
                        dummy->mNext = head2;
                        head2 = head2->mNext;
                }
                dummy = dummy->mNext;
        }
        if(NULL!=head1) dummy->mNext = head1;
        if(NULL!=head2) dummy->mNext = head2;

        return dummyNode.mNext;
}

void BucketSort(int n,int arr[]){
        vector<ListNode*> buckets(BUCKET_NUM,(ListNode*)(0));
        for(int i=0;i<n;++i){
                int index = arr[i]/BUCKET_NUM;
                ListNode *head = buckets.at(index);
                buckets.at(index) = insert(head,arr[i]);
        }
        ListNode *head = buckets.at(0);
        for(int i=1;i<BUCKET_NUM;++i){
                head = Merge(head,buckets.at(i));
        }
        for(int i=0;i<n;++i){
                arr[i] = head->mData;
                head = head->mNext;
        }
}
```

10. 基數排序

```
#include<stdio.h>
#define MAX 20
//#define SHOWPASS
#define BASE 10

void print(int *a, int n) {
  int i;
  for (i = 0; i < n; i++) {
    printf("%d\t", a[i]);
  }
}

void radixsort(int *a, int n) {
  int i, b[MAX], m = a[0], exp = 1;
```

```
  for (i = 1; i < n; i++) {
    if (a[i] > m) {
      m = a[i];
    }
  }

  while (m / exp > 0) {
    int bucket[BASE] = { 0 };

    for (i = 0; i < n; i++) {
      bucket[(a[i] / exp) % BASE]++;
    }

    for (i = 1; i < BASE; i++) {
      bucket[i] += bucket[i - 1];
    }

    for (i = n - 1; i >= 0; i--) {
      b[--bucket[(a[i] / exp) % BASE]] = a[i];
    }

    for (i = 0; i < n; i++) {
      a[i] = b[i];
    }

    exp *= BASE;

#ifdef SHOWPASS
    printf("\nPASS   : ");
    print(a, n);
#endif
  }
}

int main() {
  int arr[MAX];
  int i, n;

  printf("Enter total elements (n <= %d) : ", MAX);
  scanf("%d", &n);
  n = n < MAX ? n : MAX;

  printf("Enter %d Elements : ", n);
  for (i = 0; i < n; i++) {
    scanf("%d", &arr[i]);
  }

  printf("\nARRAY  : ");
  print(&arr[0], n);

  radixsort(&arr[0], n);
```

```
  printf("\nSORTED : ");
  print(&arr[0], n);
  printf("\n");

  return 0;
}
```