理论考试复习

- 考试题型：判断，单选，填空，程序填空

数据类型

- 基本类型：整型（short, int, long, long long），浮点型(float, double)，字符型(char)
  - 注意有无符号（unsigned）
- 常量
  - 注意整数的十进制、八进制和十六进制的表示形式
    - 十进制（%d），八进制（%o），十六进制（%x）
    - `0x`表示十六进制，`0`表示8进制
  - 注意科学计数法的表示方法：1e9，1e-9
  - 注意义转字符：`\\`，`%%`，`\n`等
  - 注意特殊的符号常量：NULL = 0，EOF = -1
  - 知道常量的定义方式：const + 数据类型 + 变量名 = 常量值
- 数组
  - 知道如何使用一维数组，二维数组
  - 能看懂数组初始化的方法：`int a[3][2] = {1, 2, 3, 4, 5, 6};`
  - 理解数组名与指针的关系
  - 能看懂strlen，strcmp，strcpy, strcat函数，并理解他们的用法
- 指针
  - 知道指针是什么：一个指向内存的变量
  - 知道*，->，& 等符号的含义
  - 特别注意指针相关的优先级：`*p++`，`(*p)++`，`++*p`
- 结构体
  - 知道什么是结构体：可以理解为多个变量构成的集合

```
struct node{
    int x, y;
};
```

  - 结构体在完成定义后，本身的使用与int等基础类型差不多
  - 知道用`.`访问结构体中的元素：`a.x`
  - 结构体数组：与基础类型的数组类似
  - 结构体指针
    - p指向结构体
    - 学会使用`->`
    - `(*p).x = p->x`
  - 能看懂结构体的嵌套
    - 多个`.`或者`->`进行访问

```
struct point{
    int x, y;
};

struct node{
    int val;
    struct point* pos;
};

struct node a;
a.pos.x;
a.pos.y;

struct node* p;
p->pos->x;
p->pos->y;
```

- 知道变量的作用范围
  - 作用域为这个变量所在的大括号内部
  - 全局变量作用范围为整个文件
  - 注意static用法：函数体中static定义的变量只会在第一次被初始化，在每次函数调用时都保留上次函数结束时的值

基本运算

- 包括算术运算（含自增、自减操作）、关系运算、逻辑运算、条件运算、赋值运算、位运算
  - `+, -, *, /, %, ++, --`
  - `> < <= >= !=`
  - `&& || !`
  - `?:`
  - `<< >> <<= >>= & | ~`
- 掌握运算符的功能
- **知道运算符的优先级！**

语句控制

- if else语句，switch语句
  - 特别去注意switch语句的用法

```
switch(expression){
    case constant-expression  :
       statement(s);
       break; /* 可选的 */
    case constant-expression  :
       statement(s);
       break; /* 可选的 */

    /* 您可以有任意数量的 case 语句 */
    default : /* 可选的 */
       statement(s);
}
```

        - **switch** 语句中的 **expression** 是一个常量表达式，必须是一个整型或枚举类型
        - case 的 **constant-expression** 必须与 switch 中的变量具有相同的数据类型，且必须是一个常量或字面量
        - 当被测试的变量等于 case 中的常量时，case 后跟的语句将被执行，直到遇到 **break** 语句为止
        - 不是每一个 case 都需要包含 **break**。如果 case 语句不包含 **break**，控制流将会 *继续* 后续的 case，直到遇到 break 为止
        - 一个 **switch** 语句可以有一个可选的 **default** case，出现在 switch 的结尾。default case 可用于在上面所有 case 都不为真时执行一个任务。default case 中的 **break** 语句不是必需的
    - 注意=与==，考试可能会挖坑！
- while, do while, for
    - 注意while与do while的区别
        - while先判断再做，do while先做再判断
    - 注意for语句结束后循环变量的值

```
for (i = 1; i <= 10; i ++ );
循环结束后i=11
```

    - 改变控制流的语句：continue，break，return
        - continue：跳到下一次循环
        - break：跳出这重循环
        - return：结束当前函数

一些常用的函数

- math.h
    - cos, **sqrt**, abs, fabs, pow, log, log10
- string.h
    - strcpy, strcmp, strcat, strlen

自定义函数的编写

```
 ------------>函数类型
 |    -------->函数名
 |    |        ------>形参表
 |    |        |
int add(int a, int b) {
    return a + b; /*函数体，写在一对大括号里*/
}            |
             |
             ------>return类型与函数类型一致

int main() {
    int x = 1, y = 2;
    printf("%d", add(x, y)); /*调用add函数*/
    return 0;
}
```

- 内容本身并不复杂，搞清楚程序的执行流程即可
- 注意区分形参和实参
- 注意变量的生命周期
- 注意static

define相关

- 注意define是直接替换，不是函数

```
#include <stdio.h>
#define f(x) x+x

int main() {
   int x = 5;
   printf("%d", f(x) * f(x)); //x*x + 2*x
   return 0;
}
```

文件

- 标准输入输出：scanf, printf, getchar, putchar, gets, puts
- 文件读写的步骤
  - 创建文件指针

    ```
    FILE *fp;
    ```

  - 打开文件

    ```
    fp = fopen("a.txt", "r");
    //fopen失败则会返回一个NULL值
    ```

- 文件处理：`fscanf、fprintf`
- 关闭文件：`fclose(fp);`

- **口 文件打开方式参数表**

文 本 文 件 (ASCII)		二 进 制 文 件(Binary)	
使用方式	含 义	使用方式	含 义
“ r ”	打开只读文件	“ rb ”	打开只读文件
“ w ”	建立只写新文件	“ wb ”	建立只写新文件
“ a ”	打开添加写文件	“ ab ”	打开添加写文件
“ r+ ”	打开读/写文件	“ rb+ ”	打开读/写文件
“ w +”	建立读/写新文件	“ wb+ ”	建立读/写新文件
“ a +”	打开读/写文件	“ ab+ ”	打开读/写文件

- 注意事项
  - 读文件时指定的文件必须存在，否则会出错
  - 写文件时
    - 若以"w"方式写，若原文件存在，则会将原文件删除重新建立；若原文件不存在，则创建该文件
    - 若以"a"方式写，若原文件存在，则写入的内容添加到原有数据后；若原文件不存在，则创建该文件
- 文件读写函数
  - 字符读写：`fgetc(), fputc()`

```
ch = fgetc(fp);
fputc(ch, fp); //成功返回ch，失败返回EOF(-1)
//getchar()
//putchar()
```

  - 字符串读写：`fgets(), fputs()`

```
fgets(s, n, fp);
//n为指定读入字符串的长度，最多读入n-1个字符，因为末尾要放'\0'
//成功返回读取的字符串，失败返回空指针
fputs(s, fp); //成功返回所写的最后一个字符，失败返回EOF
//gets(s);
//puts(s);
```

  - 格式化读写：`fscanf(), fprintf()`
  - 二进制读写：`fread(), fwrite()`
  - 判断文件是否结尾：`feof()`

```
feof(fp);
//1：文件结束，0：文件未结束
```

  - 文件定位函数`fseek(), rewind(), ftell()`

```
rewind(fp); //使文件指针指向读写文件的首地址
fseek(fp, offset, from);
//对文件指针进行移动，移动到from+offset的位置，from为SEEK_SET(0)，SEEK_CUR(1)或者SEEK_END(2)
ftell(fp);
//获取当前文件指针的位置（相对于文件开头的字节数）
```

- 检测文件读写出错函数`ferror()`
- 清除末尾标志和出错标志函数`clearerr()`