2021.1.10

1. c/c++中字符串处理

<string.h>文件

常用函数：

- strlen(字符串数组名/指针名)
- strcat(字符串数组名/指针名, 字符串数组名/指针名)
- strcpy(字符串数组名/指针名, 字符串数组名/指针名)

数组名和指针名都尝试写点

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

// 字符串指针
int main(){
    char *str1;
    // 在给指针变量赋值前，一定要先分配内存
    str1 = (char *)malloc(sizeof(char) * 10);
    strcpy(str1, "刘晗煜");
    printf("%s\n", str1);
    return 0;
}
```

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

// 字符串数组名
int main(){
    char str1[20];
    strcpy(str1, "刘晗煜");
    printf("%s\n", str1);
    return 0;
}
```

2. 什么时候可以用 char str[]

1. 数组作为形参

```c
void func(char name[]);
```

2. 数组赋值

```c
char str[] = {'l', 'i', 'u'};
```

3. 数组在结构体内（**非常不推荐，会乱码**）

```c
struct studentInfo{
    char name[];
    int id;
}
```

推荐使用：

```
#define maxn 20

struct studentInfo(){
    int id;
    char gender;
    char name[maxn];
    char major[maxn];
}Alice, Bob, stu[100];


// 定义结构体以及结构体数组
studentInfo Alice;
studentInfo stu[100];
```

3. 结构体

结构体注意事项

1. 结构体内部有： ①变量声明 ②构造函数
2. 结构体内部声明字符串时，尽量用字符串数组名形式，而不是指针(内存分配问题)

2021.1.14

1. vscode快捷键

```
# 选中部分左移一个tab
command + [

# 选中部分右移一个tab
command + ]

# (取消)窗口最大化
command + control + f

# 上一步
command + z

# 下一步
command + y
```

2021.1.15

1. git远程操作

```
# 添加远程仓库
# e.g. git remote add pyCEML git@github.com:Dirtyworker-
k2/pyCEML.git
>>> git remote add <resposity_name> <url>

>>>git remote -v
pyCEML  git@github.com:Dirtyworker-k2/pyCEML.git (fetch)
pyCEML  git@github.com:Dirtyworker-k2/pyCEML.git (push)

# 推送到远程仓库
# 当你想分享你的项目时，必须将其推送到上游。 这个命令很简单：git push
<remote> <branch>。 当你想要将 master 分支推送到 origin 服务器时（再次说
明，克隆时通常会自动帮你设置好那两个名字）， 那么运行这个命令就可以将你所做的备份
到服务器：
# e.g. git push pyCEML master
# e.g. git push pyCEML dirty
>>> git push <remote>/<resposity_name> <branch>
```

2. git提交代码到远程仓库main分支

```
# 添加远程仓库
>>> git remote add <resposity_name> <url>

# 推送到远程对应的<branch>分支, 并不是main分支
>>> git push <resposity_name> <branch>


# 需要合并master(<branch>)到main分支
>>> git fetch <resposity_name>
>>> git checkout main
>>> git merge master --allow-unrelated-histories (合并分支解决冲突)
>>> git push <resposity_name> main
```

3. git push <resposity_name> main可能遇到的问题

```
To github.com:Dirtyworker-k2/CEProcessScript.git
 ! [rejected]        main -> main (non-fast-forward)
error: 推送一些引用到 'github.com:Dirtyworker-k2/CEProcessScript.git'
失败
提示：更新被拒绝，因为您当前分支的最新提交落后于其对应的远程分支。
提示：再次推送前，先与远程变更合并（如 'git pull ...'）。详见
提示：'git push --help' 中的 'Note about fast-forwards' 小节。
```

引起这个问题是远程仓库和本地不同步引起的。 解决方案：需要先获取远端更新并与本地合并,再git push 具体操作如下:

```
>>> git pull origin main --allow-unrelated-histories
>>> git push origin main
```

2021.1.23

1. 结构体

1.1 结构体的定义与声明

两种声明方式：

```
#define maxn 20

struct studentInfo(){
    int id;
    char gender;
    char name[maxn];
    char major[maxn];
}Alice, Bob, stu[100];


// 声明结构体以及结构体数组
studentInfo Alice;
studentInfo stu[100];
```

1.2 访问结构体元素

三种访问结构体元素的方式： Alice是变量，Bob和Jony是指针

1. Alice.id
2. Bob->gender
3. (*Jony).major

```c
#include <stdio.h>

#include <stdlib.h>
#include <string.h>
#define maxn 20

struct studentInfo{
    int id;
    char gender;
    char name[maxn];
    char major[maxn];
};


int main(){
    // “.”操作访问结构体元素
    studentInfo Alice;
    Alice.id = 100;
    Alice.gender = 'f';
    printf("%d\n", Alice.id);

    // “->”操作访问结构体元素
    studentInfo *Bob;
    // 在给指针变量赋值前，一定要分配内存。否则若进入系统工作区，会引起混乱(segmentation fault)
    Bob = (studentInfo *)malloc(sizeof(studentInfo));
    Bob->id = 200;
    Bob->gender = 'm';
    printf("%c\n", Bob->gender);
    free(Bob);

    // “.”操作访问结构体元素
    studentInfo *Jony;
    Jony = (studentInfo *)malloc(sizeof(studentInfo));
    Jony->id = 300;
    strcpy(Jony->major, "Material");
    printf("%s\n", (*Jony).major);

    return 0;
}
```

1.3 结构体的初始化

法一

```
stu.id = 100
stu.gender = 'f'
```

法二

```
scanf("%d %c %s", &stu.id, &stu.gender, stu.major)
```

法三: 构造函数--用来初始化结构体的一种函数

构造函数特点：

- 不需要写返回类型
- 函数名与结构体相同

默认的构造函数：studentInfo(){}

由于这个构造函数的存在，才可以直接定义studentInfo类型的变量而不提供初始化参数

```
struct studentInfo{
    int id;
    char gender;

    // 默认的构造函数
    studentInfo(){}
};
```

如果想要提供初始化参数，则如下：

```
struct studentInfo{
    int id;
    char gender;

    // 提供id和gender初始化参数的构造函数
    studentInfo(int _id, char _gender){
        id = _id;
        gender = _gender;
    }
};

int main(){
    // 字符用'x', 字符串用"xxx"
    studentInfo me(111, 'm');
    printf("%d\n", me.id);
}
```

注意：若没有提供参数对应的构造函数，则会出现错误--error: no matching constructor for initialization of 'studentInfo'

1.4 结构体的使用

```
#include <stdio.h>
struct Point{
    int x, y;

    Point(){};

    Point(int _x, int _y){
        x = _x;
        y = _y;
    }
}pt[10];

int main(){
    int num=0;
    for (int i=0; i<3; i++){
        for (int j=0; j<3; j++){
            pt[num++] = Point(i, j);
        }
    }

    for (int i=0; i<num; i++){
        printf("(%d, %d)\n", pt[i].x, pt[i].y);
    }

    return 0;
}
```

2. 引用

2.1

函数的参数是局部变量，若想改变传入函数的参数，则需使用指针/引用

引用：不产生副本，而是给原变量起了个**别名**

```c
#include <stdio.h>

// 声明函数
void change(int &x);
void change1(int x);

int main(){
    int x=10;
    change1(x);
    printf("After change1: %d\n", x);
    change(x);
    printf("After change: %d\n", x);
    return 0;
}

// 定义函数
void change(int &x){
    x = 1;
}

void change1(int x){
    x = 1;
}
```

2.2 指针的引用

见3.2.1 错误写法二

3. 使用指针变量作为函数参数(地址传递)--区别于值传递

3.1 地址传递举例

```c
#include <stdio.h>

void change(int *p){
    *p = 233;
}

int main(){
    int a=1;
    int *x = &a;
    change(x);
    printf("%d\n", *x);
    return 0;
}
```

3.2 a、b交换值

3.2.1 地址传递

```c
#include <stdio.h>

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

int main(){
    int a=1, b=2;
    swap(&a, &b);
    printf("After change, a=%d, b=%d\n", a, b);
    return 0;
}
```

错误写法一：

```
void swap(int *a, int *b){
    int *temp;
    // 先给指针分配内存空间再赋值，否则随机地址指向系统工作区会报错
    *temp = *a;
    *a = *b;
    *b = *temp;
}
```

错误写法二：

```
void swap(int *a, int *b){
    // 类似于地址的值传递：指针变量本身的修改无法作用到原指针变量上
    int *temp = a;
    a = b;
    b = temp;
}
```

错误写法二修改： **指针的引用**

```
#include <stdio.h>

void swap(int* &p1, int* &p2){
    int *temp = p1;
    p1 = p2;
    p2 = temp;
}

int main(){
    int a=1, b=2;
    // 引用产生的是变量的别名，因此常量不可以直接使用引用
    int *p1 = &a, *p2 = &b;
    swap(p1, p2);
    printf("After change: a=%d, b=%d\n", *p1, *p2);
    return 0;
}
```

3.2.2 引用

```c
#include <stdio.h>

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

int main(){
    int a=1, b=2;
    swap(a, b);
    printf("After change, a=%d, b=%d\n", a, b);
    return 0;
}
```