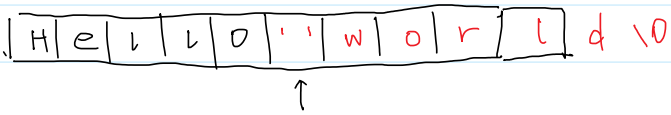
字符串

2022年3月21日 14:46

④ char * strcat (char *dest, const char* src);

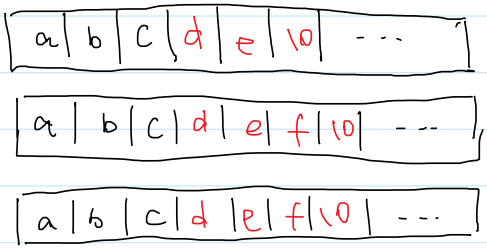
把字符串src的内容追加到字符串dest的末尾，并返回dest. (不会检查数组越界)

```
char s1[10] = "Hello";
strcat (s1, " world");
```



char *strncat (char *dest, const char *src, size_t count);

```
char s1[10] = "abc";
strncat (s1, "def", 2);
strncat (s1, "def", 3);
strncat (s1, "def", 5);
```



因为strncat总会写入'\0'，所以我们一般会这样调用

strncat (s1, s2, sizeof(s1) - strlen(s1) - 1)

↳ 预留空间给空字符

字符串的惯用法

2022年3月21日 15:00

练习1：自己编写strlen函数

```c
size_t my_strlen(const char* s) {
    size_t n;
    for (int n = 0; *s != '\0'; s++) {
        n++;
    }
    return n;
}
```

⇒

```c
size_t my_strlen(const char* s) {
    char* p = s;
    while (*p) {
        p++;
    }
    return p - s;
}
```

惯用法：搜索字符串的末尾

```c
while (*s) {
    s++;
}
```

s指向了空字符

```c
while (*s++)
    ;
```

s指向了空字符后面的字符

练习2：自己编写strcat函数

```c
char* my_strcat(char* s1, const char* s2) {
    char* p = s1;
    while (*p) {
        p++;
    }
    while (*s2 != '\0') {
        *p = *s2;
        p++;
        s2++;
    }
    *p = '\0';
    return s1;
}
```

⇒

```c
char* my_strcat(char* s1, const char* s2) {
    char* p = s1;
    while (*p) {
        p++;
    }
    while (*p++ = *s2++)
        ;
    return s1;
}
```

惯用法：复制字符串，包括空字符

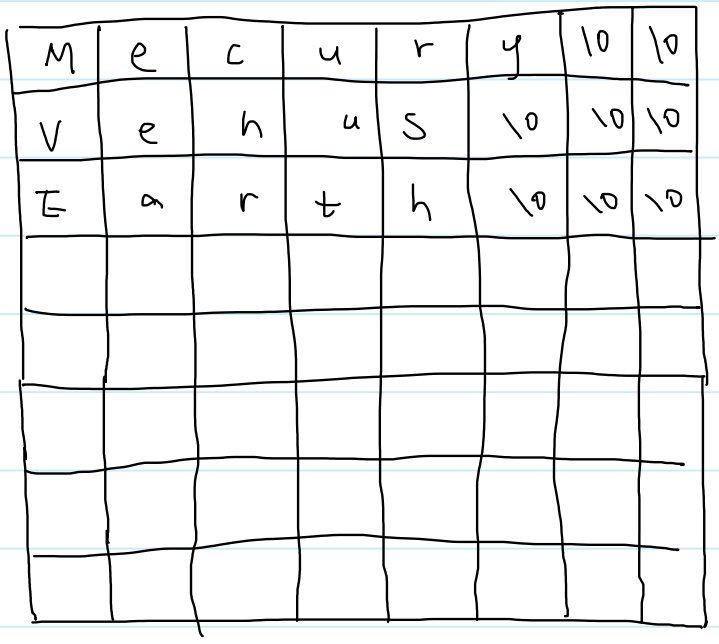
```c
while (*p++ = *s++)
    ;
```

字符串数组

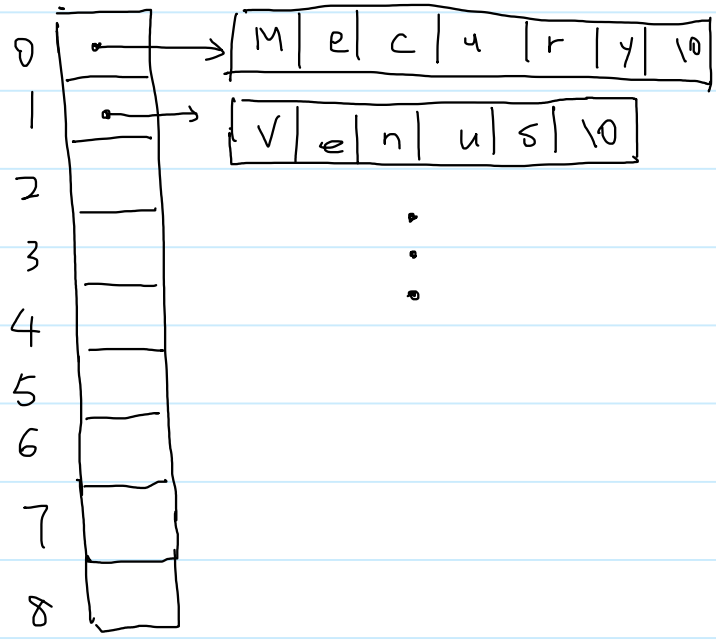
2022年3月21日 15:19

Q: 如何表示字符串数组?

```
// 二维数组
char planets[][8] = { "Mecury", "Venus", "Earth", "Mars", "Jupitor", "Saturn",
                      "Uranus", "Neptune", "Pluto" };
```




```
// 字符指针数组（推荐使用）
char* planets[] = { "Mecury", "Venus", "Earth", "Mars", "Jupitor", "Saturn",
                    "Uranus", "Neptune", "Pluto" };
```



命令行参数

2022年3月21日 15:45

程序开始，操作系统调用 main 函数

程序结束：main 返回（exit）

操作系统 —参数→ 程序？（√）

程序 —状态码→ 操作系统

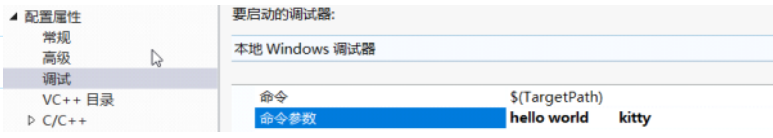
```c
int main(int argc, char* argv[]) {
    printf("argc = %d\n", argc);

    for (int i = 0; i < argc; i++) {
        puts(argv[i]);
    }

    return 0;
}
```

```
Microsoft Visual Studio 调试控制台
argc = 1
D:\code\c\43CPP\C_Day07\x64\Debug\02_命令行参数.exe
```

项目 → 属性 → 调试 → 命令参数



我们也可以通过命令行启动程序：

```
D:\code\c\43CPP\C_Day07\x64\Debug>D:\code\c\43CPP\C_Day07\x64\Debug\02_命令行参数.exe hello world kitty
argc = 4
D:\code\c\43CPP\C_Day07\x64\Debug\02_命令行参数.exe
hello
world
kitty
```

↓

第一个参数：可执行程序的路径

结构体（★★★★★）

2022年3月21日 16:00

C语言的结构体相当于其它高级语言中的类，C语言只能在结构体中定义数据.

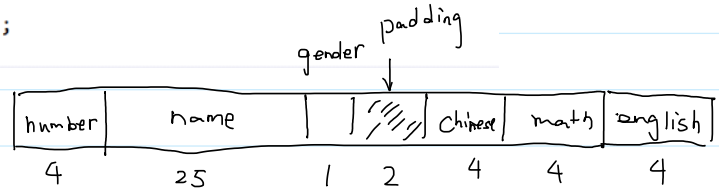
Q. 如何表示一个学生对象？

属性：学号，姓名，性别，语文成绩，数学成绩，英语成绩...

```
struct student_s {
    int number;
    char name[25];
    bool gender;   // true --> male, false --> female
    int chinese;
    int math;
    int english;
};
```

```
struct student_s s1; // 创建两个学生对象
struct student_s s2;
```

内存布局：



4 + 25 + 1 + 2 + 4 + 4 + 4 = 44

+填充的目的是为了对齐.

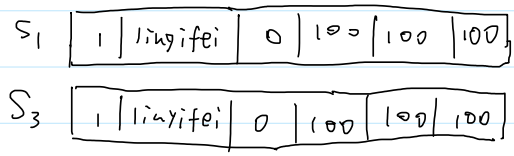
(1) 结构体对象的初始化（和数组类似）

```
struct student_s s1 = {1, "liuyifei", false, 100, 100, 100};
struct student_s s2 = {2, "huasheng", true};  → 未初始化的成员会被默认值为0
```

(2) 对结构体对操作.

① 获取成员：s_1.name

② 赋值：$s_3 = s_1$



注意事项：当结构体作为参数或者返回值时会拷贝整个结构体中的数据.

```
void print_student(struct student_s s) {   → 值传递
    printf("%d %s %d %d %d %d\n", s.number, s.name, s.gender, s.chinese, s.math, s.english);
    // s.number = 100;
}
```

为了避免拷贝数据，我们往往会传递一个指向结构体的指针

```
void print_student(struct student_s* s) {
    printf("%d %s %d %d %d %d\n", (*s).number, (*s).name, (*s).gender, (*s).chinese, (*s).math,
        (*s).english);
}
```

C语言程序一般是通过指针去引用结构体的，C语言提供了一个特别的运算符 →

```
void print_student(struct student_s* s) {
    printf("%d %s %d %d %d %d\n", s->number, s->name, s->gender, s->chinese, s->math,
        s->english);
}
```

s->name ⟺ (*s).name

(3) 使用typedef为结构体取别名

```
typedef struct student_s {
    int number;
    char name[25];
    bool gender;   // true --> male, false --> female
    int chinese;
    int math;
    int english;
} Student;
```

枚举

2022年3月21日 17:00

在程序有一些变量只能取一些离散的值. 如扑克牌的花色.

一种做法, 定义一些宏.

```
#define SUIT int
#define SPADE 0
#define HEART 1
#define CLUB  2
#define DIAMOND 3

int main(void) {
    SUIT suit = SPADE;
}
```

但更好的做法是定义枚举类型.

```
enum suit {SPADE, HEART, CLUB, DIAMOND};

int main(void) {
    enum suit s = SPADE;
}
```

```
typedef enum suit {
    SPADE,   0
    HEART,   1
    CLUB,    2
    DIAMOND  3
} Suit;

int main(void) {
    Suit s = SPADE;
    return 0;
}
```

枚举类型的值, 本质上都是一些整数值

SPADE	SPADE (0)
HEART	HEART (1)
CLUB	CLUB (2)
DIAMOND	DIAMOND (3)

当然, 我们可以指定枚举常量的值.

```
typedef enum suit {
    SPADE,
    HEART = 7,
    CLUB,
    DIAMOND = 16
} Suit;
```

SPADE	SPADE (0)
HEART	HEART (7)
CLUB	CLUB (8)
DIAMOND	DIAMOND (16)

指针的高级应用

2022年3月21日 17:14

#1 动态内存分配（重点）

动态内存分配在C语言中有着举足轻重的地位，因为它是链式结构的基础。

#2. 指向指针的指针（二级指针）

#3. 指向函数的指针（函数指针）

动态内存分配

2022年3月21日 17:19

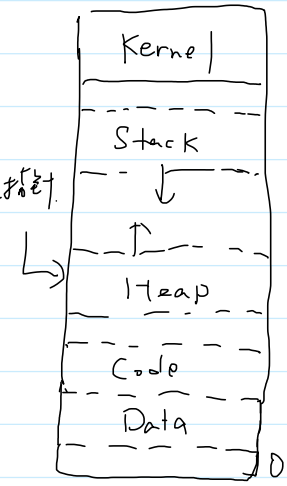
在头文件 <stdlib.h> 定义了三个动态内存分配的函数（在堆上分配内存空间）

通用指针 allocate

① void * malloc (size_t size);

↓ memory

分配 size 个字节的内存，不会内存块清零；若分配不成功，返回空指针



② void *calloc (size_t num, size_t size);

为 num 个元素分配内存空间，每个元素的大小为 size 个字节，并对内存块清零. 若分配不成功，返回空指针.

③ void *realloc (void *ptr, size_t new_size);

调整先前分配内存块的大小，如果分配成功，返回指向新内存的指针，否则返回空指针.

注意事项 ptr 应该指向先前使用动态内存分配函数分配的内存块

空指针：不指向任何对象的指针（用宏 NULL，其值为 0）

（回顾：什么是野指针）

小试牛刀. 编写一个函数，把两个字符串拼接起来，而不改变其中任何一个字符串

```c
char* my_strcat(const char* s1, const char* s2);

int main(void) {
    char* s1 = "Hello ";
    char* s2 = "world.";
    char* s = my_strcat(s1, s2);
    puts(s1);
    puts(s2);
    puts(s);

    return 0;
}

char* my_strcat(const char* s1, const char* s2) {
    char* s = (char *)malloc(strlen(s1) + strlen(s2) + 1);
    if (s == NULL) {
        // 做错误处理
        return NULL;
    }
    strcpy(s, s1);
    strcat(s, s2);
    return s;
}
```

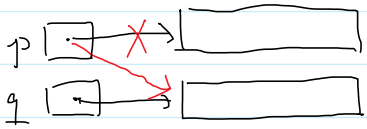
→ 这个示例并不规范，因为申请空间没有释放！

(2) 释放内存空间

如果申请的内存空间没有释放，就可能造成内存泄漏现象.

举个例子：

```
p = malloc(...);
q = malloc(...);
p = q;
```



（不能再访问的内存，垃圾）

(内存泄漏：如果程序中存在此场景，这种现象我们称为内存泄漏)

如何避免内存泄漏？答：及时释放无用的内存块

void free (void * ptr);

↳ ptr必须是先前调用malloc、calloc、realloc返回的指针.