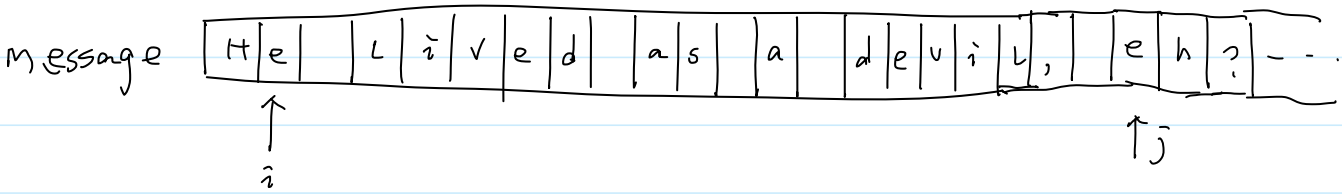
作业讲解

2024年4月30日 11:14

```
bool is_palindrome(char message[], int n) {
    int i = 0, j = n - 1;
    while (i < j) {
        while (i < j && !isalpha(message[i])) {
            i++;
        }   // i == j || message[i] is a alphabet

        while (i < j && !isalpha(message[j])) {
            j--;
        }   // i == j || message[j] is a alphabet

        if (tolower(message[i]) != tolower(message[j])) return false;

        i++;
        j--;
    }
    return true;
}
```

```
Enter a message: He lived as a devil, eh?
Palindrome
```



字符串数组

2024年4月30日 11:34

字符串数组，字符数组的数组 → 二维字符数组

数组初始化式

```
char planets[][8] = { "Mercury", "Venus", "Earth", "Mars",
                      "Jupiter", "Saturn", "Uranus", "Neptune" };
```

→ 多种方法

存储特点：

① 浪费内存空间

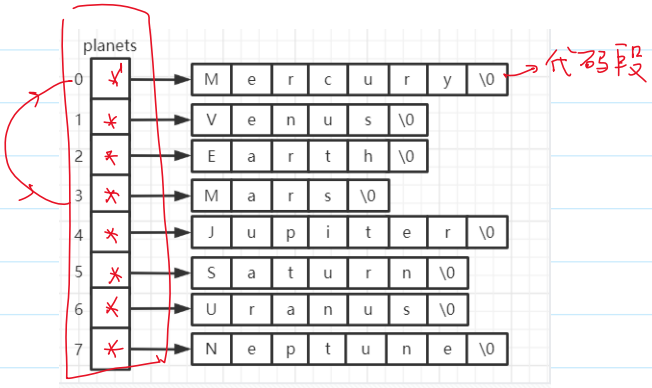
② 交换字符串不方便

	0	1	2	3	4	5	6	7
0	M	e	r	c	u	r	y	\0
1	V	e	n	u	s	\0	\0	\0
2	E	a	r	t	h	\0	\0	\0
3	M	a	r	s	\0	\0	\0	\0
4	J	u	p	i	t	e	r	\0
5	S	a	t	u	r	n	\0	\0
6	U	r	a	n	u	s	\0	\0
7	N	e	p	t	u	n	e	\0

(2) 字符指针数组 (灵活)

字面值

```
char* planets[] = {"Mercury", "Venus", "Earth", "Mars",
                   "Jupiter", "Saturn", "Uranus", "Neptune"};
```



命令行参数

2024年4月30日 11:53

操作系统 —参数→ main　命令行参数 what.

操作系统 ←状态码— main

状态码

0：成功

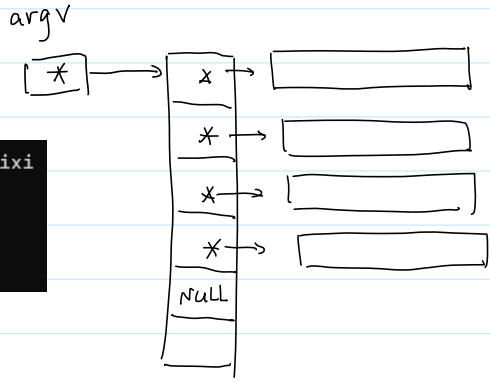
非0：失败

```c
int main(int argc, char* argv[]) {
    // argc: argument count, 命令行参数的个数
    // argv: argument vector, 命令行参数, 字符串
    //       第一个参数是可执行程序的路径

    printf("argc = %d\n", argc);
    // 打印所有的命令行参数
    for (int i = 0; i < argc; i++) {
        puts(argv[i]);
    }

    return 0;
}
```

```
D:\工作\cpp58\1_C\01_命令行参数\Debug> 01_命令行参数.exe
argc = 1
01_命令行参数.exe
```

```
D:\工作\cpp58\1_C\01_命令行参数\Debug> 01_命令行参数.exe I love xixi
argc = 4
01_命令行参数.exe
I
love
xixi
```

命令行参数



Q1. 命令行参数 和 从 stdin 读取数据 有什么区别？

命令行参数 ⇓ 程序还未执行

stdin 读取数据 ⇓ 程序已经运行

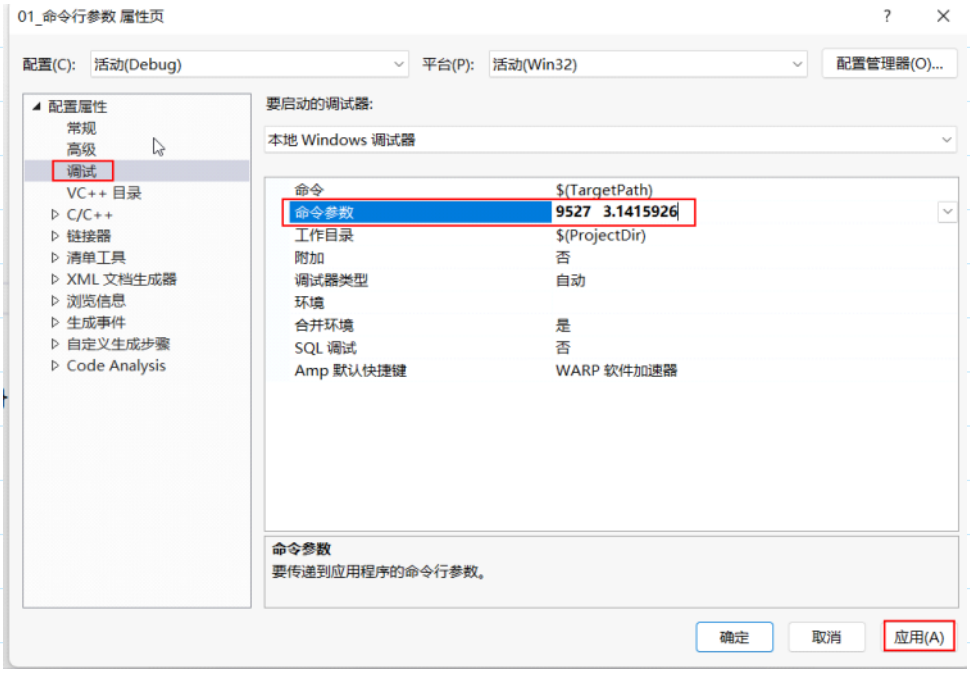
Q2. 命令行参数有什么作用.

cp src dst : 写一些非常通用程序

ls -l : 改变程序的行为，参数不同，行为不同

} why

如何在 VS 中设置命令行参数



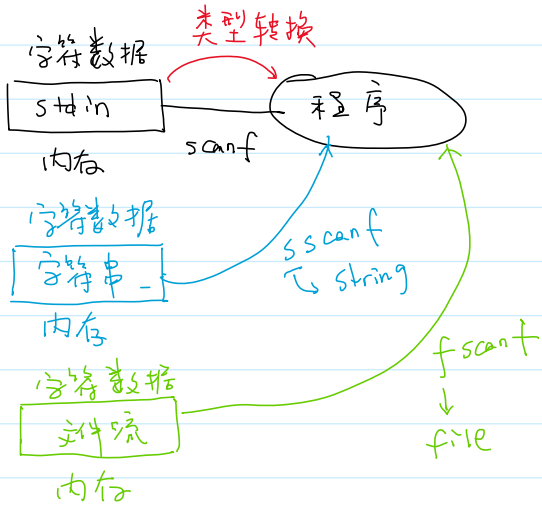

```c
// 命令行参数的转换
// xxx.exe n f
int n;
float f;

sscanf(argv[1], "%d", &n);
sscanf(argv[2], "%f", &f);

return 0;
}
```

112 % 未找到相关问题 行: 24 字符: 1

监视 1

搜索(Ctrl+E) 搜索深度: 3

名称	值	类型
n	-858993460	int
f	-107374176.	float
argv[1]	0x0133c008 "9527"	查看 char *
argv[2]	0x0133c00d "3.1415926"	查看 char *
添加要监视的项		



结构体

2024年4月30日 15:05

对象：

属性：静态数据

方法：行为

C语言的结构体类似其他语言中的"类"，不一样的是，

结构体只有属性，没有方法。（C++）

语法：

```c
// 定义结构体类型，自定义类型
struct student {
    int id;
    char name[25];
    char gender;
    int chinese;
    int math;
    int english;
};

int main(void) {
    // 声明并初始化变量
    struct student s1 = {1, "xixi", 'F', 100, 100, 100};
    struct student s2 = {2, "peanut", 'M'};
    return 0;
}
```

4
25
1
4
4
4

42

44

有2个字节的填充

未指定的成员，会赋0值。

内存模型：

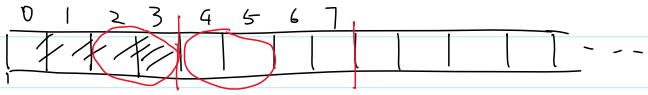
1. 一片连续的内存空间
2. 会按声明的顺序存放每一个成员
3. 在结构体变量的中间或后面，可能会有填充

学生对象的内存布局如下：

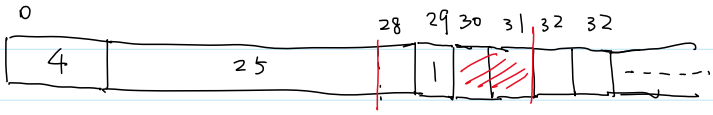
填充是为了对齐

对齐是为了更快地访问数据。

数据总共：32bit



why？

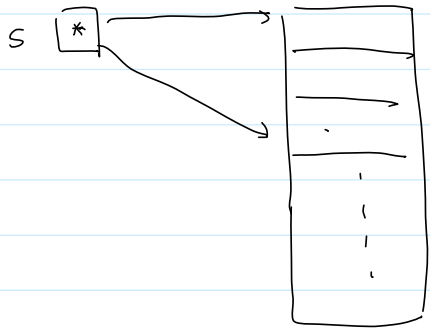


结构体变量的操作

2024年4月30日 15:51

```c
// 1. 获取成员: .
//printf("%d %s %c %d %d %d\n",
//  s1.id,
//  s1.name,
//  s1.gender,
//  s1.chinese,
//  s1.math,
//  s1.english);

// 2. 赋值: 结构体变量的复制
// 往往更习惯传递一个指向结构体变量的指针
// s2 = s1;
```

```c
void print_stu_info(const struct student* s) {
    printf("%d %s %c %d %d %d\n",
        (*s).id,
        (*s).name,
        (*s).gender,
        (*s).chinese,
        (*s).math,
        (*s).english);
}
```




```c
// 语法糖: ->
printf("%d %s %c %d %d %d\n",
    s->id,
    s->name,
    s->gender,
    s->chinese,
    s->math,
    s->english);
```

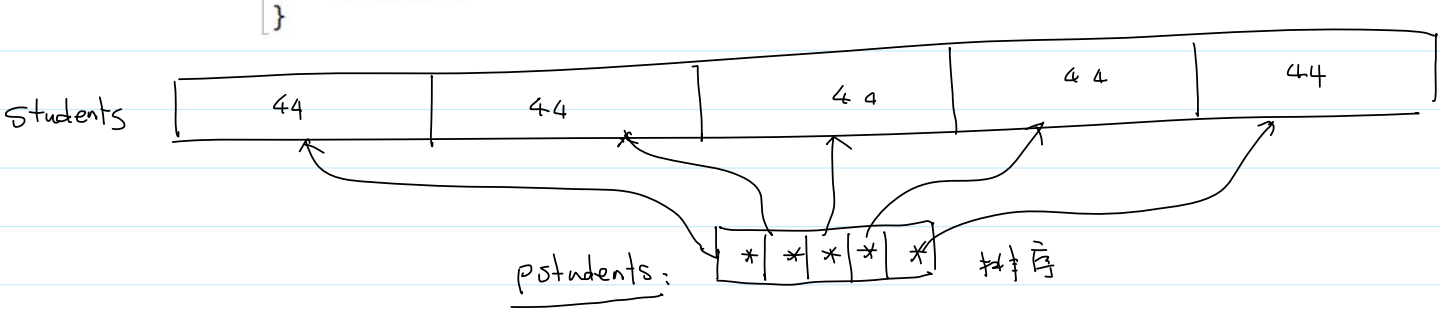
右箭头运算符：左边，指针

右边，成员

课堂小练习

2024年4月30日 16:06

```c
int main(void) {
    Student students[5];
    for (int i = 0; i < 5; i++) {
        scanf("%d%s %c%d%d%d",
            &students[i].id,
            students[i].name,
            &students[i].gender,
            &students[i].chinese,
            &students[i].math,
            &students[i].english);
    }

    Student* pstudents[5] = {students, students + 1, students + 2,
        students + 3, students + 4};

    return 0;
}
```



枚举

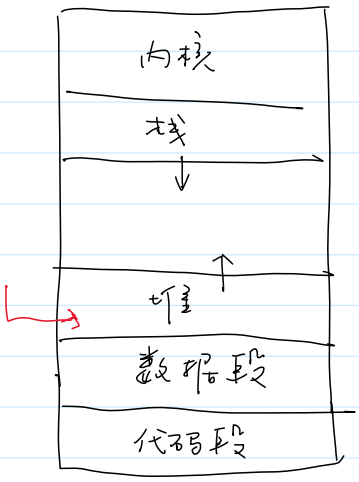
2024年4月30日 17:02

作用，表示一些离散值

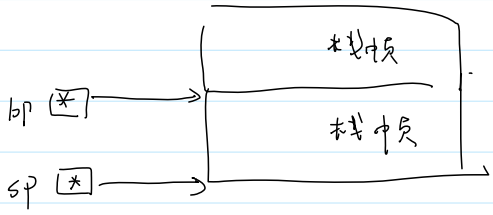
类别、状态.

动态内存分配（~~~~~）

2024年4月30日 17:16



free list
伙伴算法



Q1：为什么要在堆上分配空间？

⇓

栈空间受限的

A. 栈帧的大小是在编译期间确定的

栈只能存放动态大小的数据

B. 栈空间比较小。

主线程。8M

其他线程。2M

栈上不能存很大的数据

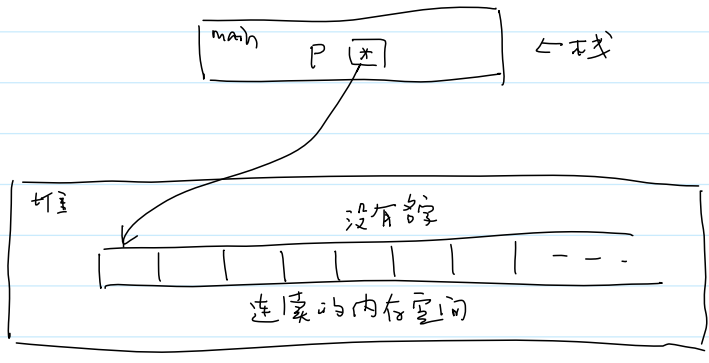
C。栈空间最好不要被多线程共享的东西

#2. How 如何在申请堆空间。

↗ memory + allocate

- `void* malloc(size_t size)`
  分配 size 个字节的内存块，不对内存块进行清零；如果无法分配指定大小的内存块，返回空指针。
- `void* calloc(size_t nmemb, size_t size)` → clear + allocate
  为有 nmemb 个元素的数组分配内存块，其中每个元素占 size 个字节，并且对内存块进行清零；如果无法分配指定大小的内存块，返回空指针。
- `void* realloc(void *ptr, size_t size)`
  调整先前分配内存块的大小。如果重新分配内存大小成功，返回指向新内存块的指针，否则返回空指针。

```c
int main(void) {
    int* p = malloc(sizeof(int) * 100);

    return 0;
}
```



void*：通用指针

作用：可以和其它类型指针相互转换，

指向对象的类型还不确定，

↓

不能够直接操作通用指针。

晚自习答疑

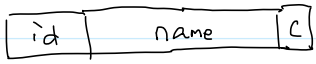
2024年4月30日 21:29

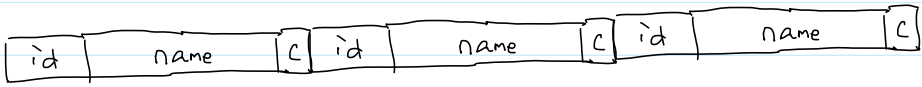
Struct Foo;

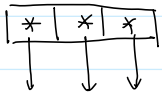
Struct Foo *

Struct Foo arr[3];

Struct Foo* arr[3]







Student * pstudents [5];

void average (Student* S, int n);

average (pstudents, 5);