C语言入门（*CS100*）Part Five

> *-- By Yiming Li*

> 本文章的内容参考至*Stephen Prata*的《C Primer Plus》

> 本内容适用于*CS*专业学生

字符串

- 使用双引号括起来的都是字符串
- 标准表示方法：`char str[4] = {'a','b','c','\0'};`,或者是`char str[4] = "abc";`,而字符串的指针的表示方法就是`char *arr = "abc";`。
- 注意这里的指针和平常所说的首地址不一样
    1. `char arr[] = "abc";`分配一个固定大小的数组来储存这些字符串，字符串"abc"被复制到这个数组中。数组可以在程序中修改。数组名是这个数组首元素的地址
    2. `char *arr = "abc";`只分配一个指针变量来储存字符串的地址，字符串`"abc"`储存在**只读的内存空间（通常是常量池），不能通过指针修改该这一些内容**
    3. 指针可以重新指向其它字符串或地址
- 举个例子

```c
#include <stdio.h>
int main()
{
   char str[4] = "abc";
   printf("%s\n",str1);
   return 0;
}
```

- 在这个例子中我们就定义好了一个字符串，但是有几个小细节需要注意一下：
    1. 在底层，实际储存的时候，C语言还是会帮我们将"abc"转换成为字符数组进行保存，并且末尾还要加上`\0`。
    2. 数组的长度要么不写，如果要写的话一定要预留结束空间的长度
    3. 使用字符数组和双引号的方式来定义的字符串的内容是可以改变的

```c
//方式二
#include <stdio.h>
int main()
{
    char *str1 = "abc";
    printf("%s\n", str1);
}
```

- 几个细节：

    1. 在底层，实际储存的时候，C语言还是会将字符串"abc"转换成字符数组进行保存，并且在末尾还要加上\0.
    2. 利用指针+双引号的方式定义的字符串，会把底层的字符数组放在只读常量区（常量池）
- 只读常量区的特点
    1. 内容不可以进行修改
    2. 里面定义的字符串是可以重复使用的

```c
#include <stdio.h>
int main()
{
    char *str1 = "abc";
    char *str2 = "abc";
    printf("%s\n", str1);
    printf("%p\t%p\n", str1, str2);
}
```

```
abc
00007FF67ADD5050        00007FF67ADD5050
```

- 从下面的例子来看这两个地址相同，你可以看出来如果两个都是只读的字符串，那么在存放的时候会比较两者是否是相同的，如果是相同的字符串，那么就会**复用**。（注意不能够写成printf("%p",&str1);这句话表示的就是指针的地址）

```c
// 使用键盘输入一个字符串并打印
#include <stdio.h>
int main()
{
    char str[100];
    int len = sizeof(str) / sizeof(char);
    scanf("%s", str);
    for (int i = 0; i < len; i++)
    {
        if (str[i] == '\0')
            break;
        else
            printf("%c", str[i]);
    }
    printf("'\\0'");
    return 0;
}
```

```
liym2024@shanghaitech.edu.cn
liym2024@shanghaitech.edu.cn'\0'
```

- 不能够通过指针的方式来进行录入，因为不能够进行修改

字符串数组

- 本质：就是一个二维数组
- 定义
  - 第一种方法

```c
#include <stdio.h>
int main()
{
   char str1[] = "liym2024@shanghaitech.edu.cn";
   char str2[] = "3987412763@qq.com";
   char str3[] = "17775756985";
   char *str[] = {str1, str2, str3};
   int len[] = {sizeof(str1), sizeof(str2), sizeof(str3)};
   for (int i = 0; i < 3; i++)
   {
      // 复制指针，避免修改原始指针
      char *ptr = str[i];
      for (int j = 0; j < len[i]; j++)
      {
            printf("%c", *ptr);
            ptr++;
      }
      printf("\n");
   }
   return 0;
}
```

  - 特点：

1. **定义字符数组**：
   - `char str1[]`, `char str2[]`, `char str3[]` 分别定义了三个字符数组，每个数组都包含一个字符串。
2. **指针数组初始化**：
   - `char *str[] = {str1, str2, str3};` 定义了一个指针数组，每个元素是指向上述字符数组的指针。
3. **长度计算**：
   - `int len[] = {sizeof(str1), sizeof(str2), sizeof(str3)};` 使用 `sizeof` 来获取每个字符数组的大小（包括终止符 `\0`），这并不是字符串的实际长度。
4. **遍历和打印**：

        - 使用双重循环逐个字符地打印每个字符串。外层循环遍历指针数组，内层循环通过解引用指针并递增指针来逐个字符打印。
    - 第二种方法

```c
#include <stdio.h>
int main()
{
    char *str[3] = {
        "liym2024@shanghaitech.edu.cn",
        "3987412763@qq.com",
        "17775756985"};
    for (int i = 0; i < 3; i++)
    {
        char *strx = str[i];
        printf("%s\n", strx);
    }
    return 0;
}
```

    - 主要区别

1. **字符串存储位置**：
    - **第一种方法**：字符串存储在可写的字符数组中，可以通过指针修改内容（尽管在这个例子中没有修改）。
    - **第二种方法**：字符串存储在只读内存区域中，不能通过指针修改内容。
2. **遍历方式**：
    - **第一种方法**：逐个字符遍历和打印，适用于需要对字符串进行逐字符处理的场景。
    - **第二种方法**：直接打印整个字符串，适用于只需要输出字符串的场景。
3. **灵活性**：
    - **第一种方法**：可以修改字符数组中的内容，适合需要动态修改字符串的场景。
    - **第二种方法**：不能修改字符串内容，但更简洁高效。

字符串的常见函数

- `strlen()`:获取长度
- `strcat`:拼接两个字符串
- `strcpy`:复制字符串
- `strcmp`:比较两个字符串
- `strlwr`:将字符串变成小写
- `strupr`:将字符串变成大写

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

```c
int main()
{
    char *str1 = "abc"; // 中文占据三个字节
    char str2[100] = "ABASDFK@#^&*ALSJDFZXAFSDG";
    char str3[5] = {'q', 'w', 'e', 'r', '\0'};
    printf("-------------------strlen()----------------------\n");
    int len1 = strlen(str1);
    int len2 = strlen(str2);
    int len3 = strlen(str3);
    printf("%d,%d,%d\n", len1, len2, len3); // strlen在统计长度的时候不算'\0'
    printf("-------------------strcat()----------------------\n");
    strcat(str2, str3); // 放到了末尾
    /*
        细节1：第一个字符串是可以被修改的
        细节2：第一个字符串中剩余空间可以容纳拼接后的字符串
    */
    printf("%s\n", str2);
    printf("%s\n", str3);
    printf("-------------------strcpy()----------------------\n");
    strcpy(str1, str3);
    // 将第二个字符串的内容拷贝到第一个字符串中，并且将第一个字符串中的内容给覆盖掉了
    /*
        细节1：第一个字符串是可以被修改的
        细节2：第一个字符串中所有空间可以容纳拷贝的字符串
    */
    printf("%s\n", str2); // qwer
    printf("%s\n", str3); // qwer
    printf("-----------------strcmp()----------------------\n");
    int res = strcmp(str1, str2);//完全一样0,不一非零
    printf("%d\n", res);
    printf("-----------------_strlwr()----------------------\n");
    _strlwr(str2); // 只能够转换英文的大小写
    printf("%s\n", str2);
    printf("-----------------_strupr()----------------------\n");
    _strupr(str2);
    printf("%s\n", str2);
}
```

结构体

- 结构体可以理解为自定义的数据类型，他是由一批数据组合成为的结构性数据。

```c
struct 结构体名称
{
   成员1;
   成员2;
   ...
}
```

```c
#include <stdio.h>
#include <string.h>
struct GirlFriend
{
    char name[100];
    int age;
    char gender;
    double height;
};
struct Student
{
   char name[30];
   int age;
   double height;
}
int main()
{
    /*
        结构体:
            自定义的数据类型
            就是由很多的数据组合成为的一个整体
            每一个数据，都是结构体的成员
        书写的位置:
            函数里面：局部位置只能在本函数中使用
            函数外面：在所有的函数中都可以使用
    */
   // 使用结构体
   // 定义一个类型的变量
   struct GirlFriend gf1;
   strcpy(gf1.name, "aaa");//字符串的赋值需要注意
   gf1.age = 21;
   gf1.gender = 'F';
   gf1.height = 1.63;
   //--------------------------------
   struct Student stu1 = {"Sam",18,175.26};
   struct Student stu2 = {"Lily",17,16.35};
   struct Student strArr[2] = {stu1,stu2};
   // 遍历每一个元素
   for(int i = 0;i < 2; i++)
   {
      struct Student temp = strArr[i];
      printf("%s %d %lf",temp.name,temp.age,temp.height);
   }
   return 0;
}
```

结构体的别名

```c
typedef struct (Name)//name可以写也可以不写
{
```

```
    char name[100];
    int age;
    char gender;
    double height;
}GF;
```

```
#include <stdio.h>
#include <string.h>
typedef struct Ultram
{
    char name[100];
    int attack;
    int defense;
    int blood;
} M;

int main()
{
    M taro = {"Laitor", 100, 90, 500};
    M rem = {"rem", 90, 80, 450};
    M arr[2] = {taro, rem};
    for (int i = 0; i < 2; i++)
    {
        M temp = arr[i];
        printf("%s %d %d %d\n", temp.name, temp.attack, temp.defense, temp.blood);
    }
    return 0;
}
```

- 相当于在这里我们将`struct Ultram`改成了`M`.

结构体作为函数的参数

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

typedef struct Student
{
    char name[30];
    int age;
} stu;
void change(stu *st);
int main(void)
{
    stu stu1;
    strcpy(stu1.name, "Liyiming");
    stu1.age = 18;
    change(&stu1);
```

```
    printf("The name of stu1 is %s\n", stu1.name);
    printf("The age of stu1 is %d\n", stu1.age);
    return 0;
}
void change(stu *st)
{
    printf("The name of stu1 is %s\n", (*st).name);
    printf("The age of stu1 is %d\n", (*st).age);
    scanf("%s", (*st).name);
    scanf("%d", &(*st).age);
}
```

结构体的嵌套

```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>
struct Message
{
    char phone[12];
    char school_mail[100];
};

typedef struct Student
{
    char name[30];
    int age;
    struct Message msg;
} stu;

void change(stu *st);
int main(void)
{
    stu stu1;
    strcpy(stu1.name, "Liyiming");
    stu1.age = 18;
    strcpy(stu1.msg.phone, "17775756985");
    strcpy(stu1.msg.school_mail, "liym2024@shanghaitech.edu.cn");
    change(&stu1);
    printf("%s", stu1.msg.school_mail);
    return 0;
    /*
    stu stu2 = {"Liyiming",18,{"17775756985","liym2024@shanghaitech.edu.cn"}};
    */
}
void change(stu *st)
{
    scanf("%s", (*st).msg.school_mail);
}
```

结构体的内存对齐

- 确定变量的位置：
    1. 总体上还是按照定义的顺序从前到后的安排内存地址
    2. 每一个变量只能放在自己类型整数倍的内存地址上（中间空出来的字节会被补位空白字符）
- 最后一个补位：结构体的总大小，是最大类型的整数倍
- 补位并不会改变相应的类型的变量的大小
- 其实不只是在结构体中，只要是储存变量就会存在内存对齐的情况
- 综上：我们将小的数据类型写在上面，大的数据类型写在下面（节省空间）