1、数据类型

表格

描述已自动生成

|  |  |
| --- | --- |
| **转义字符** | **解释** |
| \n | 换行符，表示光标移到下一行的开头 |
| \0 | 空字符，没有任何内容，字符串的结束标志 |
| \r | 回车符，表示光标移到当前行的开头 |
| \t | 制表符，相当于tab键，跳8个空 |
| \' | 用于表示字符常量' |
| \'' | 用于表示一个字符串内部的双引号 |
| \\ | 用于表示一个反斜杠，防止它被解释为转义字符 |
| \b | 退格符 |
| \f | 换页符 |

2、数据类型

|  |  |  |
| --- | --- | --- |
| **常量类型** | **说明** | **举例** |
| 字面常量 | 直接写在代码中的固定数值 | 10、3.14、'A'、”Hello” |
| 常量类型 | 说明 | 举例 |
| 字面常量 | 直接写在代码中的固定数值 | 10、3.14、'A'、“Hello” |
| 符号常量 | 通过宏定义指令定义的，用来代表一个固定的数值或字符串 | 通过 #define PI 3.14159 定义了一个符号常量 PI，可以在程序中多次使用 PI 来表示圆周率； |

3、占位符

|  |  |  |  |
| --- | --- | --- | --- |
| %d： | int， | %f： | float |
| %hd： | short， | %lf： | double |
| %ld： | long， | %c： | char |
| %lld： | long long | %s： | string |
| %p: | 指针 |  |  |

scanf("%d", &num1);

%取余数

位移指令 左乘右除2

**char str[] = "Hello,world!"; char \*str = "Hello,world!";**

表格

中度可信度描述已自动生成表格, 日历

中度可信度描述已自动生成

4、左移右移

int为有符号数 unsigned为无符号数

▪️左移操作可能会导致数据溢出；

▪️右移运算符有两种形式：算术右移和逻辑右移

算术右移：对于有符号数，算术右移会将最高位（符号位）保持不变，其他位向右移动

逻辑右移：对于无符号数，逻辑右移会将所有位都向右移动，并用0填充左侧的空位。

逻辑右移不考虑符号位。

▪️如果右移的位数大于等于操作数的位数，结果将为0。另外，右移操作可能会导致数据丢失；

▪️移位运算符通常用于底层编程、位操作、加密算法等领域，需要熟练掌握它们的使用。

|  |  |
| --- | --- |
| **操作符** | **解释说明** |
| 按位与：& | 对两个二进制对应位都为1时，结果为1，否则为0 |
| 按位或：｜ | 对两个二进制对应位只要有一个为1时，结果为1，否则为0 |
| 按位取反：～ | 对一个二进制数的每一位进行取反操作，将0变为1，将1变为0 |
| 按位异或：^ | 对两个二进制数的每一位进行异或操作，当两个对应位不相同时，结果为1，否则为0。 |

|  |  |  |
| --- | --- | --- |
| **优先级** | **操作符** | **解释** |
| 1 | ()、 []、 ->、 . | 括号、数组、两种结构成员访问 |
| 2 | !、~、++、--、+、-、\*、&、（类型）、sizeof | 否定、按位取反、自增、自减、正号、负号、间接访问、地址访问、类型转换、内存大小 |
| 3 | \* 、/、% | 乘、除、取余 |
| 4 | ‘+、- | 加、减 |
| 5 | <<、>> | 左移、右移 |
| 6 | <、<=、>=、> | 小于、小于等于、大于等于、大于 |
| 7 | ==、!= | 等于、不等于 |
| 8 | & | 按位与 |
| 9 | ^ | 按位异或 |
| 10 | ｜ | 按位或 |
| 11 | && | 逻辑与 |
| 12 | || | 逻辑或 |
| 13 | ? : | 条件操作符 |
| 14 | ‘=、+=、-=、\*=、/=、%= | 赋值、加等、减等、乘等、除等、取模等 |
| 15 | ， | 逗号运算 |

文本

描述已自动生成

文本

描述已自动生成文本

描述已自动生成

图示, 示意图

描述已自动生成图片包含 文本

描述已自动生成

随机数生成

#include <stdlib.h>

#include <time.h>

    srand(time(NULL)); // 随机数种子,不使用time(NULL)的话每次运行程序，随机数种子都一样，产生随机数序列也都一样

    int rn = rand() % 100 + 1; // rand() % 100 + 1 产生1-100之间的随机数

//计算公式：如果要产生[A-B]之间的随机整数，则：rand() % (B - A + 1) + A

#include <string.h>

char \*strcpy(char \*dest, const char \*src);//字符串赋值函数

int strcmp(const char \*str1, const char \*str2);==0时为相同

strcasecmp()//忽略大小写

atoi将字符串转换为数值

冒泡

图示

描述已自动生成文本

描述已自动生成

图片包含 文本

描述已自动生成

获取字符串长度

strlen(str)

文本

描述已自动生成

\*：间接访问操作符(指针标记)；

&：地址运算符；

指针=首地址

printf("%p\n",ptr); //打印地址

指针名可以当做数组名使用

漏斗图

描述已自动生成

int 每步移动4格

long long 每步移动8格

char 每步移动1格

两个指针相差的步数

指针相减运算只有在指针指向同一数组时才有意义

指针关系运算只有在指针指向同一数组元素时才有意义

二级指针

指针函数

指针函数可以用来返回动态分配的内存地址，这样可以在函数外部访问和操作该内存，灵活地处理数据。

#include <stdlib.h>

int \*fun2(){

    int \*p = (int \*)malloc(40\*sizeof(int));//malloc动态申请内存，分配的是堆空间,申请的内存不会随着函数的调用结束而自动释放，必须手动释放，否则会造成内存泄露

    //malloc函数申请的内存，必须free释放，否则会造成内存泄露

    return p;//返回的是地址，所以可以访问局部变量

}

    //释放内存

    free(q);

    q = NULL;//释放后，指针置为空

函数指针

函数指针则可以用于实现回调函数，通过将函数指针作为参数传递给其他函数，实现在不同函数之间灵活调用和切换功能。

int add\_sub(int (\**f*)(int, int), int *a*, int *b*){//f是函数指针，指向一个函数，回调函数

    return *f*(*a*,*b*);

}

快速排序规则函数

int cmp(const void \**a*, const void \**b*){

   //return \*(int \*)a - \*(int \*)b;//升序

   return \*(int \*)*b* - \*(int \*)*a*;//降序

}//float函数为><号，int型为-号

**结构体**

结构体指针（同一般指针）

struct Student \* ptr = &s;

结构体自引用

struct Node {

int data;//数据域

struct Node\* next;//指针域

};

结构体传参

文本

描述已自动生成

**内存对齐的目的：**提高内存访问效率和处理器的数据访问速度

**枚举**

enum{

枚举值；

}

联合体（尽量别用）

union 联合名称{

类型 成员变量；

}

**file类型**

int main() {

FILE\* file\_ptr = NULL;

return 0;

}

**打开文件**

FILE\* fopen(const char\* **filename**, const char\* **mode**);

图示

中度可信度描述已自动生成

**关闭文件**

int fclose(FILE\* filename);



**行缓冲:**

标准IO库函数，往标准输出(屏幕)输出东西的时候是行缓冲的.

所谓的行缓冲就是缓冲区碰到换行符的时候才刷新缓冲区.

刷新缓冲区方法1 使用\n

**printf**("hello world\n");

刷新缓冲区方法2 程序正常结束 return 0;

**printf**("hello world");

    return 0;

刷新缓冲区方法3 使用fflush函数刷新缓冲区

**printf**("hello world");

**fflush**(stdout);

默认行缓冲区1024个字节,当缓冲区满的时候自动刷新.

    int i;

    for(i=1;i<300;i++){

**printf**("%03d ",i);

    }

**C语言中有三个特殊的文件指针无需定义**，在程序中可以直接使用。

stdin :标准输入，默认为当前终端(键盘)

scanf,getchar函数默认从此终端获得数据

stdout : 标准输出,默认为当前终端(屏幕)

printf,puts函数默认输出信息到此终端

stderr : 标准错误输出设备文件，默认为当前终端(屏幕)

当我们程序出错使用: perror函数时信息打印在此终端

Int count=**fread**(buffer,100,4,fp)

Fp指向文件的内容读取到buffer的内存中，读取的字节数为，每块100个字节，4块

如果读取到400个字节返回count 为4

如果读到了>=300 <400 返回3

如果读到了>=200 <300 返回2

如果读到了>=100 <200 返回1

**标准函数库函数**

|  |  |  |  |
| --- | --- | --- | --- |
| **功能** | **函数原型** | **返回值** | **注意事项** |
| 打开文件 | **fopen**(const char\* filename, const char\* mode) | 文件地址(NULL) |  |
| 关闭文件 | **fclose**(FILE\* filename) | 无 |  |
| 字符读取 | **fgetc**(FILE \*stream) | ASCII码(EOF-1) |  |
| 字符写入 | **fputc**(int char, FILE \*stream) | ASCII码(EOF-1) |  |
| 字符串读取 | **fgets**(char \*str, int n, FILE \*stream) | 地址(NULL) | 每次读取指定数量的字符，实际读入的字符个数等于指定的数量-1，并主动在最后添加'\0'。每次只能读取一行字符，如果需要连续读取多行字符，可以使用循环结构。 |
| 字符串写入 | **fputs**(const char \*str, FILE \*stream) | 非负值(EOF-1) | 每次写入一行文本数据到文件，不会自动添加换行 |
| 格式化读取 | **fscanf**(FILE \*stream, const char \*format, ...) | 成功读取的数据项数(EOF-1) | 在读取过程中遇到无法匹配的数据项或者读取错误，会停止读取后续数据项，直接返回当前已成功读取的数据项数 |
| 格式化写入 | **fprintf**(FILE \*stream, const char \*format, ...) |  | fprintf可以把非文本数据（格式化数据）转为文本数据，然后在写入文件，并不能写入非文本数据。 |
| 数据块读取 | **fread**(void \*ptr, size\_t size, size\_t nmemb, FILE \*stream) | 实际返回块数 |  |
| 数据块写入 | **fwrite**(const void \*ptr, size\_t size, size\_t nmemb, FILE \*stream) |  |  |
| 比较函数 | **char \*strstr**(const char \*haystack, const char \*needle) | 返回第一个出现的指针地址 | **（）** |
|  |  |  |  |
| 重置指针 | **rewind**（FILE\* ptr） |  |  |
| 指针定位 | **int fseek**(FILE \*stream, long int offset, int whence) | 0(非0) | 常用的whence参数有SEEK\_SET（从文件开头开始偏移）、SEEK\_CUR（从当前位置开始偏移）和SEEK\_END（从文件末尾开始偏移）。 |

**文件操作模式**

|  |  |  |
| --- | --- | --- |
| **文件使用方式** | **含义** | **如果指定文件不存在** |
| **“r”（只读）** | 为了输入数据，打开一个已经存在的文本文件 | 出错 |
| **“w”（只写）** | 为了输出数据，打开一个文本文件 | 建立一个新的文件 |
| **“a”（追加）** | 向文本文件尾添加数据 | 建立一个新的文件 |
| “rb”（只读） | 为了输入数据，打开一个二进制文件 | 出错 |
| “wb”（只写） | 为了输出数据，打开一个二进制文件 | 建立一个新的文件 |
| “ab”（追加） | 向一个二进制文件尾添加数据 | 出错 |
| “r+”（读写） | 为了读和写，打开一个文本文件 | 出错 |
| “w+”（读写） | 为了读和写，建议一个新的文件 | 建立一个新的文件 |
| “a+”（读写） | 打开一个文件，在文件尾进行读写 | 建立一个新的文件 |
| “rb+”（读写） | 为了读和写打开一个二进制文件 | 出错 |
| “wb+”（读写） | 为了读和写，新建一个新的二进制文件 | 建立一个新的文件 |
| “ab+”（读写） | 打开一个二进制文件，在文件尾进行读和写 | 建立一个新的文件 |

|  |  |  |
| --- | --- | --- |
| **读写方式** | **应用场景** | **举例** |
| 字符 | 处理单个字符的场景 | 键盘读取 |
| 文本行 | 处理文本文件的场景 | 处理配置文件、日志文件、文本文档等 |
| 格式化 | 处理具有特定格式的数据的场景 | 比如将一个整数格式化为字符串、将字符串解析为浮点数等 |
| 数据块 | 处理二进制数据的场景 | 处理图像、音频、视频等二进制文件，以及网络传输中的数据块 |

**动态内存**

**内存区域:**

代码区: 存放程序代码和各个函数

数据区: 存放程序中的全局变量和静态变量，常量

栈区: 存放局部变量，形参

堆区: 存放动态数据，需要用指针访问（FILE\*）

**内存泄漏:**

申请的内存，首地址丢了，找不到了，再也没法使用，也无法释放了，这块内存就被泄漏了.

#include <stdio.h>

int **main**(){

    char \*P;

    p=(char \*)malloc(100);

    p="hello";

return 0;

//free(p)

}

**malloc函数**

在堆上分配，返回首地址，int \* ptr =(int\*)malloc(sizeof(int));

**calloc 函数**

在堆上分配，返回首地址，同时初始化为0，int\* ptr=(int\*)calloc(5,sizeof(int));

**realloc 函数**

重新分配已经分配的内存块大小，ptr=(int\*)realloc(ptr,10\*sizeof(int));

**alloc 函数**

在栈上分配，返回首地址，int\* ptr=(int\*)alloca(5\*sizeof(int));

**获取时间**

#include <stdio.h>

#include <time.h>

int main() {

    time\_t now;

    struct tm \*local;

    time(&now);  // 获取当前时间

    local = localtime(&now);  // 转换为本地时间

printf("Current local time and date: %s", asctime(local));

Sun Jun 23 21:39:42 2024

    printf("Year: %d\n", local->tm\_year + 1900); // 年份是从1900年开始的

    printf("Month: %d\n", local->tm\_mon + 1);    // 月份从0开始，所以加1

    printf("Day: %d\n", local->tm\_mday);

    printf("Hour: %d\n", local->tm\_hour);

    printf("Minute: %d\n", local->tm\_min);

    printf("Second: %d\n", local->tm\_sec);

    return 0;

}

struct tm {

  int tm\_sec;   // 秒，范围从 0 到 59

  int tm\_min;   // 分，范围从 0 到 59

  int tm\_hour;  // 时，范围从 0 到 23

  int tm\_mday;  // 一个月中的日期，范围从 1 到 31

  int tm\_mon;   // 月份，范围从 0 到 11

  int tm\_year;  // 自 1900 年起的年数

  int tm\_wday;  // 一周中的日期，范围从 0（星期日）到 6（星期六）

  int tm\_yday;  // 一年中的日期，范围从 0 到 365

  int tm\_isdst; // 夏令时标识符

};