C

一种基于C语言的helloWorld

```c
#include<stdio.h>
int main(){
    printf("hello world");
    return 0;
}
```

- 整数类型%d：

1. int，整数，4
2. short，占据空间更小的整数，2
3. long，长整数，8
4. unsigned int，正整数，4
5. unsigned short，占据空间较小的正整数，2
6. unsigned long，长正整数，8

- 小数类型%f：

1. float，小数，4
2. double，更长的小数，8
3. long double，非常长的小数，16

- ASCII码%c：

  char，-127~127，1

- 自动类型转换

  字节数较小的数据可以向字节数较大的进行转换:
  char, short -> int -> unsigned -> long -> double <- float

- 强制转换：（类型）表达式 (int)(10+' a'+i*f-d/e)

输入输出

第一行输入 #include<stdio.h> 来引入如下函数

1. 字符输入：getchar，字符输出“putchar
2. 格式输入：scanf，格式输出：printf
3. 字符串输入：gets，字符串输出：puts

常见字符输出类型：%d，%f，%c，%s

顺序结构，选择结构

优先级：（*，/），（%），（+，-）

注意！ 在表达式中，++和--如果在后面，则会在表达式计算完成后再计算++或--；反之则会优先计算

```
int a = 4, b = 5;
int s;
s = a++ + b;    //此时s=9, a=5, b=5
s = ++a + b;    //此时s=11, b=6, b=5
```

- 与&&, 或||, 非!
- 条件运算

```
// 如果表达式1==True, x=表达式2, 否则x=表达式3
x = (表达式1) ? (表达式2) : (表达式3)
```

数组

```
//下面语句中, a[0]=1, a[1]=3, a[2]=5, a[3]=0
int a[4] = {1, 3, 5};

//下面语句相当于 int a[3] = {1, 3, 5}
int a[] = {1, 3, 5};
```

函数

参数

- 函数名括号内的变量成为“形式参数”（简称“形参”）
- 在其它函数内调用函数，则括号内变量称为“实际参数”（“实参”）
- 函数间的数据传递分为值传递和地址传递：
    - 值传递：实参单向传递值给形参（它们的类型必须相同）
    - 地址传递：数据存储地址作为参数双向传递给形参（形参和实参必须暂用相同的存储单元，并且必须是地址常量或变量）
    - 如果实参和形参都是数组，则是地址传递

返回类型

- 在定义函数时需要定义它对应返回值的类型
- 如果是void，则不能出现return，并且在该函数内的功能不会影响函数外的参数

递归

- 函数直接或间接调用自己，称为递归
- 递归三要素：
    1. 初始条件

2. 转移条件
3. 终止条件

```c
/*一个递归的例子*/
#include<stdio.h>

void inversion(int n){  //初始条件
    int t;
    if(n > 10){ //终止条件
        t = n%10;
        printf("%d", t);
        inversion(n/10);    //转移条件
    } else {
    printf("%d", n);
    }
}

int main(){
    int a;
    scanf("%d", &a);
    inversion(a);
    printf("\n");
    return 0;
}
```

预编译

- 对命令进行预处理，将其结果和源程序一起进行编译处理，得到目标代码（OBJ文件）。

  1. 宏命令（Macro）
  2. 文件包含命令（include）
  3. 条件编译命令

  这些命令均以#开头

  ```c
  //不带参数的宏
  #define pi 3.14
  //终止宏定义
  #undef pi

  //文件包含
  #include " (文件名) "
  ```

指针

内存是按照地址来访问的，地址指是一个个编号，也就是指针（相当于内存被分成一个个小格子）

- 变量地址：系统分配给变量的内存单元的起始地址

- 利用变量名，直接存取变量值，成为“直接访问”
- 定义一个待存放地址的指针：【 数据类型 *变量名; 】

```
int a=1, b=2, c, *pc;
pc = &c;    //将c的地址赋给pc，此时pc所存放的就是c的地址
*pc = a+b;    //此时c = a+b = 3
```

由上可知：

*变量名 表示指针变量，它等于其指向地址所存储的数值
&变量名 表示该变量的地址，通常为16进制整数，输出时用%p

对于上式，*pc = c = 3，pc = &c = c的地址（指针的变量只能保存地址量）

- 指针必须绑定数据类型
- 对于 *&变量名 ，其含义为 先取得变量名的地址，再进行*运算。*&a和a等价
- 仅进行地址交换时，不会影响数值；仅进行数值交换时，不会影响地址

指针和数组

将数组起始地址赋给指针，通过该指针可以访问数组中的全部元素。

C语言中数组名代表的时数组的首地址，
因此（int *pa, int a[10]） pa = a 和 pa = &a[0] 是等价的。

指针和函数

一个用指针调用函数的例子

```
#include<stdio.h>

int max(int b[]){}  //选出数组中的最大值

int main(){
    int i, m, a[10], max(int *);
    int (*pf)() //定义指向函数的指针
    for(i=0; i<10; i++)
        scanf("%d", &a[i]);
    pf = max;   //指针初始化，将函数名赋值给指针
    m = (*pf)(a);   //通过指针来调用函数
    printf("max = %d\n", m);
    return 0;
}
```

一个用指针创建函数，并将其调用的例子

```c
#include<stdio.h>

float *search(float(*pointer)[4], int n){
    float *pt;
    pt = *(pointer + n);
    return (pt);
}

int main(){

    float score[][4] = {{60, 70, 80, 90}, {56, 89, 67, 88}, {34, 78, 90, 66}};
    float *p;
    int i, m;

    printf("输入学生序号：");
    scanf("%d", &m);
    printf("学生%d的分数如下：\n", m);
    float *search(float(*point)[4], int n);
    p = search(score, m);

    for(i=0; i<4; i++)
        printf("%f\t", *(p+i));

}
```

结构体

结构体是一种构造数据的类型，它把不同类型的数据组合成一个整体。
结构体类型定义只描述结构的组织形式，不分配内存。

```c
/*这是一个学生信息结构体的例子*/
struct std{
    int stdNum;
    char stdName[30];
    char stdSex;
    int scrEng;
    int scrMath;
    int scrPhy;
}

struct std std01={1,"Tom","m",88,89,90}, std02;     //声明变量，此时才开始分配内存
/*
声明变量也可以在定义结构体时，
在结构体末尾的【}】后面跟上变量名以创建变量
例如
struct std{
    ...
}std01,std02;
*/
```

```
std02.stdNum=2;
strcpy(std02.stdName, "Alice");     //strcpy使用时需要【#include<string.h>】
strcpy(std02.stdSex, "f");
std02.scrEng=91;
std02.scrMath=92;
std02.scrPhy=93;

struct std *pstd = &std01;
//其中一种输出方式
printf("学号%d，姓名%c，性别%c，英语成绩%d，数学成绩%d，物理成绩%d", (*pstd).stdNum,
(*pstd).name, (*pstd).stdSex, (*pstd).scrEng, (*pstd).scrMath, (*pstd).scrPhy);
//另一种输出方式
printf("学号%d，姓名%c，性别%c，英语成绩%d，数学成绩%d，物理成绩%d", pstd->stdNum,
pstd->name, pstd->stdSex, pstd->scrEng, pstd->scrMath, pstd->scrPhy);
```

结构体指针

指向结构体**变量**的指针，结构体比那辆的起始地址就是该结构体变量的指针。
当把结构体变量的地址存放在一个指针变量中，该指针变量就指向结构体变量。

单链表

动态进行存储分配的一种结构。

每个节点包含两个部分： 1. 用户需要用到实际数据 2. 下一个节点的地址

可以使用结构体创建链表

```
struct Std{
    int num;
    struct Std *next;    //next指向结构体变量
};
```

自定义类型声明 typedef

typedef 原类型名 新类型名;

它可以配合struct来自定义一个单链表类型

```
typedef struct Lnode{
    int data;    //保存节点的数值
    struct Lnode *next;      //指针域
}Lnode, *LinkList; //节点的类型，Lnode是变量，LinkList是指针
/*
在此时
Lnode *p;
则相当于
LinkList p;
*/
```

```
//给一个节点赋值
Lnode *p;
p = (Lnode*)malloc(sizeof(Lnode));  //分配内存，并将内存首地址赋给p
p -> data=20;
p -> next=NULL;
```

枚举类型

enum 枚举名{枚举元素列表};
枚举元素就是枚举常量

```
enum Weekday{sum, mon, tue, wed, thu, fri, sat};

enum Weekday workday, weekend;  //声明两个枚举变量，也可以像struct跟在【}】后面直接声明
```