2初识C语言

一. 操作符

3. 位操作符

// & – 按位与　按2进制
// ^ – 按位异或
// | – 按位或

& 只要有0就为0，两个都为1才为1
^ 相同为0，相异为1
| 只要有1个真就为真

& – 按位与

```c
int main()
{
    int a = 3;
    int b = 5;
    // 00000000000000000000000000000011   3 -> 011
    // 00000000000000000000000000000101   5 -> 101
    // & 只要有0就为0，两个都为1才为1
    // 00000000000000000000000000000001
    int c = a & b;
    printf("%d\n", c); // 1
    return 0;
}
```

^ – 按位异或

```c
int main()
{
    int a = 3;
    int b = 5;
    int c = a ^ b;
    // ^ 相同为0，相异为1
    // 00000000000000000000000000000110
    printf("%d\n", c); // 6
    return 0;
}
```

| – 按位或

```c
int main()
{
    int a = 3;
    int b = 5;
    int c = a | b;
    // | 只要有1个真就为真
    // 00000000000000000000000000000111
    printf("%d\n", c); // 7
    return 0;
}
```

4. 赋值操作符

```c
// 赋值操作符
int main()
{
    int a = 10; // 创建变量a，并初始化为0
    a = 20; // 赋值
    a += 10; // 符合赋值
    // 复合赋值符 += -= *= /= &= ^= |= >>= <<=
    return 0;
}
```

5. 单目操作符

// ! 逻辑反操作
// – 负值 + 正值
// sizeof 操作数的类型长度
// ~ 对一个数的二进制按位取反
// 前置、后置-- ++ 前置、后置++
// * 间接访问操作符(解引用操作符)
// (类型) 强制类型转换

单目操作符：只有一个操作数
a和b是+的两个操作数 +是双目操作符

真和假
0为假 非0为真

! 逻辑反操作

```c
int main()
{
    int a = 10;
    printf("%d\n", !a); // 0
    printf("%d\n", !a); // !逻辑反操作  10是真，变为假0
    return 0;
}
```

正值+ 负值–

```c
int main()
{
    int a = 10;
    int b = -a; // 负值
}
```

sizeof

// 计算的是变量/类型所占空间的大小，单位是字节

```c
int main()
{
    int a = 10;
    printf("%d\n", sizeof(a)); // 4
    printf("%d\n", sizeof(int));

    int arr[10] = { 0 }; //10个整型元素的数组
    int sz = 0;
    //计算数组的元素大小
    //个数 = 数组总大小/每个元素的大小
    sz = sizeof(arr) / sizeof(arr[0]);
    printf("sz = %d\n", sz);

    return 0;
}
```

~ 按位取反

// 对一数的二进制按位取反

```c
int main()
{
    int a = 0; // 4个字节，32bit位
    int b = ~a; // b是有符号的整形
    // 00000000000000000000000000000000
    // 11111111111111111111111111111111  有符号的整形最高位是1为负，0为正
    // 原码 反码 补码
    // 原码符号位不变，其余按位取反得反码；反码加一得补码。补码减一得反码；反码按位取反得原码。
    // ~ 按（2进制）位取反
    // 负数在内存中存储的时候，存储的是二进制的补码
    printf("%d\n", b); //使用的，打印的是这个数的原码
    // 11111111111111111111111111111111补
    // 11111111111111111111111111111110反
    // 10000000000000000000000000000001原  1为负，-1
    return 0;
}
```

数据的存储 原反补

// 整数在内存中存储的时候，存储是二进制
// 一个整数的二进制表示有3中形式:
// 原码 反码 补码
// 正的整数：原反补相同
// 负的整数：要计算
// 原码的符号位不变，其它位按位取反得反码。反码的二进制序列+1得补码
//
// 有符号的整数，最高位是0，表示正数
// 1，表示负数
//

// 内存中存储整数的时候，存储的是二进制的补码
// 计算的时候才用的也是补码

-- ++ 前置 后置

前置++

```c
int main()
{
    int a = 2;
    // a++; // a=a+1  a+=1

    // 前置++ 后置++
    int c = ++a; // 前置++  先++，后使用
    printf("c=%d\n", c);
    printf("a=%d\n", a);
    return 0;
}
```

后置++

```c
int main()
{
    int a = 2;
    int c = ++a; // 后置++  先使用，后++
    printf("c=%d\n", c);
    printf("a=%d\n", a);
    return 0;
}
```

(类型) 强制类型转换

```c
int main()
{
    int a = (int)3.14; // 3
    // 尽量避免
    return 0;
}
```

6. 关系操作符

> >= < <= != 用于测试“不相等” == 用于测试“相等”

7. 逻辑操作符

// %% – 逻辑与 – 并且 两个都满足
// || – 逻辑或 – 或者 只要有一个满足

```c
int main()
{
    int a = 3;
    int b = 5;
    if ((a == 3) && (b == 5))
    {
        printf("hehe\n");
    }

    if ((a == 3) || (b == 4))
    {
        printf("haha");
    }
    return 0;
}
```

8. 条件操作符（三目操作符）

exp1 ? exp2 : exp3;

先执行exp1；若为真，执行exp2为整个表达式的结果；若为假，执行exp3为整个表达式的结果

```c
int main()
{
    int a = 10;
    int b = 0;
    if (a == 5)
    {
        b = -6;
    }
    else
    {
        b = 6;
    }

    b = ((a == 5) ? -6 : 6);
    return 0;
}
```

9. 逗号表达式

// exp1, exp2, exp3, ...expn

// 逗号表达式会从左向右依次计算

// 整个逗号表达式的结果是最后一个表达式的结果

```c
int main()
{
    int a = 0;
    int b = 3;
    int c = -1;
    int d = (a = b - 5, b = a + c, c = a + b, c -= 5);
    printf("%d\n", d); // -10
    return 0;
}
```

10. 下标引用、函数调用和结构成员

[]　()　.　->

```c
#include <stdio.h>

int Add(int x, int y)
{
    int z = 0;
    z = x + y;
    return z;
}

int main()
{
    //int arr[10] = { 0 };
    //arr[4]; // [] - 下标引用操作符

    int a = 10;
    int b = 20;
    int sum = Add(a, b); // () - 函数调用操作符

    return 0;
}
```

二. 常用关键字

```
// 常用关键字
// auto break停止，中断（用于循环） case char const常属性 contine继续（用于循环）
// default do double else enum枚举
// extern声明外部符号 float单精度浮点数 for goto语句 if int long
// register寄存器关键字 return返回函数 short signed有符号的 sizeof计算大小
// static静态的 struct结构体关键字 switch typedef类型定义/重定义 union联合体/共用体
// unsigned void空 volatile（Linux中-易变的） while循环
// 不能与符号名冲突

//auto int a = 10; // 局部变量-自动变量 auto自动省略

// typedef - 类型定义/类型重定义    unsigned int  u_int
```

1. auto自动省略

```
//auto int a = 10; // 局部变量-自动变量
```

2. signed – 有符号的

```
// int 定义的变量是有符号的  signed int - signed省略
无符号：unsigned  ---unsigned int num = 1;无符号数
```

3. typedef – 类型定义/类型重定义

```
// unsigned int  u_int
```

4. register 寄存器关键字

```
// register int num = 10; // 建议把a定义成寄存器变量，编译器自行判断
// &num; // err 取地址取的是内存 寄存器独立于内存
// 寄存器是存储空间，在电脑上一般是集成到CPU上的，和内存是独立的存储空间
// 寄存器 高度缓存 内存 硬盘 网盘 -- 速度由高到低
```

5. static 静态的

static在C语言中的用法：

// 1. 修饰局部变量
// 2. 修试全局变量
// 3. 修饰函数

① 修饰局部变量

```
void test()
{
    int a = 1;
    // 局部变量 进创建 出销毁
    a++;
    printf("%d ", a); // 十个2
}
```

例：

```
void test()
{
    int a = 1;
    // 局部变量 进创建 出销毁
    a++;
    printf("%d ", a); // 十个2
}

int main()
{
    int i = 0;
    while (i < 10)
    {
        test();
        i++;
    }
    return 0;
}
```

// static 修饰局部变量
// 改变了变量的生命周期；不影响作用域，因为还是局部变量

```c
void test()
{
    static int a = 1;
    // static 修饰局部变量
    // 改变了变量的生命周期；不影响作用域，因为还是局部变量
    a++;
    printf("%d ", a); // 2-11
}

int main()
{
    int i = 0;
    while (i < 10)
    {
        test();
        i++;
    }
    return 0;
}
```

② 修饰全局变量 extern 声明外部符号

添加源文件add.c

```c
#define _CRT_SECURE_NO_WARNINGS 1

int g_val = 2021;
```

加上extern 就可以使用g_val变量

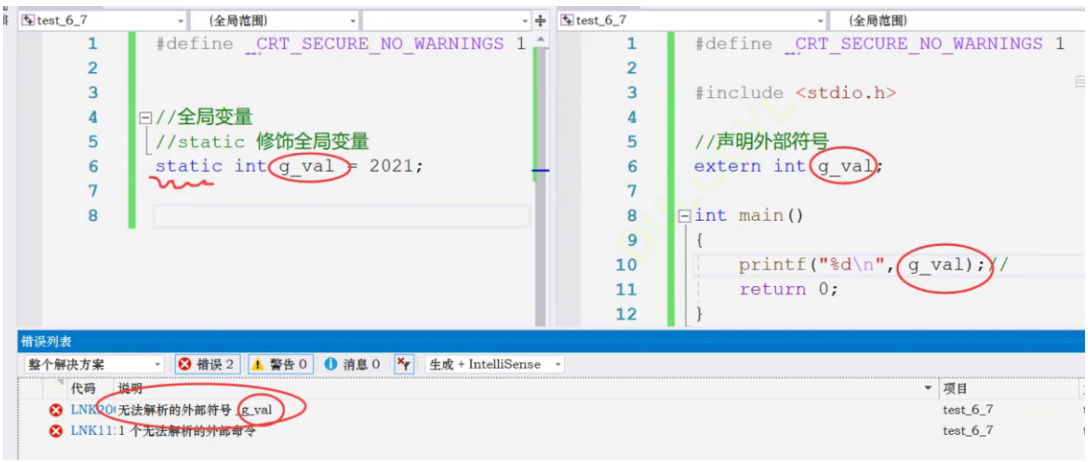
```c
extern int g_val;

int main()
{
    printf("%d\n", g_val);
    return 0;
}
```

如果加上static 就无法使用

// 默认一个全局变量是具有【外部】链接属性的

// 而如果全局变量被static修饰，全局变量的外部属性变成了内部链接属性，这个时候全局变量只能在本源文件内部使用

// 其他源文件无法链接到，也无法使用！



③ 修饰函数

```c
// 修饰函数
extern int Add(int x, int y);

int main()
{
    int a = 10;
    int b = 20;

    int c = Add(a, b);
    printf("%d\n", c);

    return 0;
}
```

```c

// 修饰函数
int g_val = 2021;

int Add(int x, int y)
{
    return x + y;
}
```

// 如果中添加static，无法使用
// 函数是具有外部链接属性，如果被static修饰，外部链接属性就变成了内部连接属性
// 函数只能在自己的源文件内部使用，不能在其他源文件内部使用！

6. #define 定义常量和宏

```c
#define NUM 100

int main()
{
    printf("%d\n", NUM);
    return 0;
}
```

MAX–宏名（X, Y)–宏变量（X>Y?X:Y)–宏主体
宏作用：替换
宏变量没有类型

```c
#define MAX(X, Y) (X>Y?X:Y)

int main()
{
    int a = 10;
    int b = 20;

    int c = MAX(a, b);
    // int c = (a > b ? a : b);
    printf("%d\n", c);

    return 0;
}
```

三 指针

内存

① 地址怎么产生：

每个内存单元都有编号
32位

32位地址线/数据线
通电转换为数字信号有正负电
正电是1负电是0
00000000000000000000000000000000
00000000000000000000000000000001
00000000000000000000000000000010
...
11111111111111111111111111111111
32个二进制序列

② 一个内存单元应该是多大的空间：字节byte

bit *8
byte *1024
kb
mb
gb
tb
pb

2^32bit =
4,294,967,296bit/8 = 536,870,912byte
= 524,288kb = 512MB = 0.5GB

如果一个内存单元是一个bit
char – 1byte – 8bit – 8个地址
int – 4byte – 32bit – 32个地址
short – 2byte – 16bit – 16个地址
太浪费了

%p – 按地址的方式打印

%s字符串 %c字符 %d整形 %p地址

```
int main()
{
    int a = 10; // 向内存申请4个字节空间，里面存放10
    printf("%p\n", &a); // & - 取地址操作符 单目

    int* pa = &a;
    // pa 是一个存放地址的变量，称为指针变量
    // int* 指针变量的类型

    char ch = 'w';
    char* pa = &ch;

    return 0;
}
```

* – 解引用操作符

```c
int main()
{
    int a = 10;
    int* pa = &a;
    *pa = 20; // * - 解引用操作符
    printf("%d\n", a);
    return 0;
}
```

```
Microsoft Visual
20

E:\bite\C\test_
按任意键关闭此窗
```

```c
int main()
{
    char ch = 'w';
    char* pc = &ch;
    *pc = 'b';
    printf("%c\n", ch);
    return 0;
}
```

指针变量的大小

```c
int main()
{
    printf("%d\n", sizeof(char*));
    printf("%d\n", sizeof(short*));
    printf("%d\n", sizeof(int*));
    printf("%d\n", sizeof(long*));
    printf("%d\n", sizeof(float*));
    printf("%d\n", sizeof(double*));

    return 0;
}
```

```
Microsoft Visual Studio 调试控
4
4
4
4
4
4

E:\bite\C\test_6_16\Debug
按任意键关闭此窗口. . .
```

64位机器 – 64根地址线

64bit – 8byte

总结：指针大小在32位平台是4个字节，64位平台是8个字节

00000000000000000000000000000000

四. 结构体

```c
struct Student
{
    char name[20];
    int age;
```

```
    char sex[5];
    char id[12];
};

int main()
{
    struct Student s1 = { "张三", 20, "男", "1905468128" };
    struct Student s1 = { "李四", 19, "女", "1904927125" };

    return 0;
}
```

打印

```
struct Book
{
    char name[20];
    int price;
    char auther;
};

int main()
{
    struct Book b1 = { "百年孤独", 55, "马尔克斯" };

    struct Book* pb = &b1;
    printf("%s %d %s\n", (*pb).name, (*pb).price, (*pb).author);
    printf("%s %d %s\n", pb->name, pb->price, pb->author);

    // . 结构体变量.成员名
    // -> 结构体->成员名
    return 0;
}
```