1. 源程序经过预处理(完成后生成.i)->编译(完成后生成.s)->汇编(完成后生成.o)->链接(完成后成为可执行文件)
2. CPU地址总线的位数(也就是电脑是32位或者是64位 说的是地址总线)决定了CPU的寻址能力，数据总线的位数决定了CPU与外界数据传输速度，控制总线的位数决定了CPU对外部部件的控制能力
3. 最小的寻址单位与内存编址单位：Byte 最小的内存单位：bit
4. 一个字节范围表示的范围[-128,127]的解释：因为这是以内存值的补码的给出的值，常识（计算机编码只能做加法运算）只有采用补码的规则才能满足计算机编码运算。
5. 打印一个字节的数的二进制：

void disBin(char n)

{

int i = 8 ;

while (i--)

{

if(n & 0x80){

printf("1");

}

else

printf("0");

n <<= 1;

}

}

1. 负整数的补码求法：将其对应的正数的二进制表示的所有位取反(包括符号位)，之后加1
2. ‘a’ 对应 ASCII为97 ‘A’对应ASCII为65 ‘\0’对应为0 ‘ ’对应为32 ‘0’对应48
3. 常量的用途，主要用于赋值，运算，常量有两种表现形式，字面的量以及宏，常量也是有类型的，比如 char name[] = “SooChow University”就是字符串常量
4. 实型常量0.0； 0.；.0；都是一样的 0.5 与.5 也是一样的； 5.0 和5.也是一样的
5. Scanf拥有天然的间隔：空格 tab enter 再enter结束输入 ，格式串与被输入的变量一一对应，非格式串需要原样输入。 对于scanf(“%c%c”)两个或者两个以上的%c连用的时候，在中间加上空格scanf(“%c %c”)，与%c相关的输入，记得留空格
6. Int类型的变量与float类型的变量在一起运算，会将int类型转换为float类型
7. 在进行运算时，以表达式中最长类型为主，将其他类型位据均转换成该类型；char short int 类型在一起运算时，首先提升到int类型
8. 隐式转化：

double tmp = 0.3;//0.3 0.7都有问题 0.3 可能在计算机中是存储的0.29999... 0.7可能是0.69999

printf("%d\n", (int)(tmp\*10));//括号的优先级 高于乘法

printf("%d\n", (int)(（tmp+0.00001(修正值)）\*10));//可以加修正值调整

1. A= B =C 是先将C赋值给B，再将整个B=C表达式的值（表达式的值等于值）赋值给A
2. A=b=c //等于号的运行顺序从右向左，相当于a=(b=c) 将b=c表达式的值赋给a
3. Printf函数的返回值，等于输出字符的个数，即格式化字符串中字符的个数

int count = printf("ni hao csdn,%d\n",111); //这里的格式化字符串包括逗号、空格、换行符、字母及数字共计16个字符

printf("n = %d\n",n);//结果为16

1. 表达式=操作数+运算符 表达式+分号构成C语言语句
2. 按操作的数的个数来分单目、双目、三目
3. 后缀++ -- 优先级高于前++ --
4. +++在编译原理中，叫大嘴原则
5. 、取余运算的符号（数字的正负），等同于被求余的数、被除数小于除数时，运算结果等于被除数 取余运算：x%y = x - y(x/y) 而且取余运算要求操作数必须是整数 例如： -7%3 的结果 首先肯定是负数，其次为什么不是-2 ，一开始我以为因为-3\*3= -9 应该取余-2，实际上应该是-7-3\*(-7/3)，-7/3的结果是-2，所以结果是-1
6. 关系运算符的结合性是从左到右
7. a>b?a:c>d?c:d; // 是等价关系a>b?a:(c>d?c:d); 探求if else 嵌套关系？
8. 逗号运算符优先级最低 ()的优先级最高，算术>关系>逻辑>条件>赋值>逗号（并非绝对 (!)） 单目运算符>双目运算符>三目运算符（并非绝对( , = )）
9. for(;;); //无条件循环
10. array 相当于 &array[0] array[0] 相当于 &array[0][0]

当array+1的时候 可以认为是 &array[0]+1 -> array[0]+1这个步长应该是数组每一列的大小；当array[0]+1的时候 可以认为是&array[0][0]+1 -> array[0][0]+1，这个步长就是基本类型的大小了。

总而言之，&（取址符号）会将修饰的变量的等级升一级，再参加运算，实际上将&去掉再看变量就会降一级。

1. 储存空间的第一个字节的地址，即低位字节的地址
2. &a 进行取地址，取出的地址是有类型的
3. type \* 变量名 type:表示变量的内存放的地址的寻址能力 \* ：表示该变量是指针变量
4. 指针变量指向谁，就是保存了谁的地址
5. NULL是一个宏，等价于(void\*)0，(void\*)是计算机黑洞，对于被赋值NULL的指针变量，不会出现内存损坏。
6. 指针变量的地址与指针变量保存的地址是不一样的
7. int \*p, q; 表示的就是定义了一个int \*型的指针变量p，一个int类型的整型变量q
8. int\* p和int \*p 含义完全一样
9. int \*ptr\_var; ptr\_var、&ptr\_var、\*ptr\_var 三者的区别？

int \*ptr\_var; //int 类型指针 ptr\_var //指针变量名 &ptr\_var//指针变量的地址 \*ptr\_var //存放在指针变量中的地址所指向的内容

1. int array[10] = {1,2,3,4,5,6,7,8,9,0};

for(int i=0; i<10; i++)

{

printf("array[%d] = %d\n",i,\*(array+i));// array+i相当于&arr[0]+i

}

int array[3][4] = {1,2,3,4,5,6,7,8,9,0};

for (int i = 0; i < 3; i++)

{

for(int j = 0; j < 4; j++)

{

printf("array[%d][%d] = %d\n", i, j, \*(array[i] + j));/\*array[i] + j相当于&array[i][0]\*/

}

}

1. a[i]相当于\*(a+i)，a[i][j]相当于\*(\*(a+i)+j) \*相当于将变量升了一级
2. arr[i] 第i 行，首元素的地址 一级指针
3. arr+i 第i 行的地址 数组指针
4. {}是作用域的限定符，{}以内的区域是局部作用域并且以内的变量都是局部变量，作用域起始于定义处，截止于所在的大括号(局部变量未赋值的话，则为随机值)，{}以外的区域是全局作用域并且以外的区域的变量都是全局变量，作用域起始于定义处，截止于本文件结束。(全局变量未赋值的话，则为0)
5. 有重名的变量时，小范围作用域的变量名会覆盖大范围作用域的变量名（局部变量同全

局变量重名的问题）

1. Extern只能用来修饰全局变量，经extern声明的变量，不可以在初始化。
2. static 修饰 局部变量，使其生命周期同进程或是main()函数。若static变量未初始化，系统初始化为0，并且只能进行一次初始化。
3. static 修饰，全局变量，限制了他的外延性。使其成为仅在本文件内部使用的全

局变量(所以一般在单文件中使用全局变量，最好使用static修饰)

1. C语言没有字符串类型，C++语言有字符类型
2. 字符’\0’,我们称为字符串结束字符，是系统对双引号引起的字符串自动加设的。
3. scanf 遇到空格会截止输入 scanf会自动在字符串后+‘\0‘
4. gets直到遇到回车，才会停止输入，空格也作为字符输入，会自动在字符串后+‘\0‘
5. 常见的栈溢出：局部变量过多，过大 或 递归层数太多。
6. 内存释放后，要将p指针置NULL，以便后续使用
7. 初始化是一种特殊语法，与赋值不等价
8. ->:指向成员运算符（相当于（\*）.） 例如：(指针变量p=&stu(结构体地址))p->name 等价于 \*(&(stu.name))。
9. struct student { char name[30]; char sex; struct birthday{

int year; int month; int day;}birth; //如果未声明birth成员，则类型只有int

1. 结构体成员内存分布，首成员在低地址，尾成员在高地址
2. struct student { char name[30]; char sex; struct birthday{

int year; int month; int day;}birth; //如果未声明birth成员，则类型只有int

1. 内存对齐 ：外对齐大小为取结构体中类型最大值m 将每一个结构体的成员大小与 Y 比较取小者为X,作为内对齐大小 所谓按 X 对齐，即为地址(设起始地址为0)能被X 整除的地方开始存放数据。 外部对齐原则是依据Y 的值(Y 的最小整数倍)，进行补空操作。
2. 共用体(union)占用内存空间的大小取决于类型长度最大的成员，union成员享有同一个起始地址。
3. 大端模式：数据的低位保存在内存的高地址，数据的高位保存在内存的低地址中
4. 小端模式：数据的高位保存在内存的高地址，数据的低位保存在内存的低地址中
5. 枚举类型(enum)定义了一组整型常量的集合，第一个枚举成员的默认值为整型0后续枚举成员的值在前一个成员上+1，可以人为设定枚举成员的值，从而自定义某个范围内的整数
6. 空链表要实现存储的自由，要动态的申请堆里的空间
7. FILE 结构体是对缓冲区和文件读写状态的记录者，所有对文件的操作，都是通过FILE 结构体完成的
8. 异或位运算：在某些位保持不变的情况下，将其余位取反
9. 隐式转化:混合提升（当有unsigned int类型数据）
10. int **main**()

{

//结果不正常

unsigned int a = 1;

int b = -100;

printf("a +b = %u\n",a+b);//混合提升 将int类型的数据 转换为了unsigned int

printf("a +b = %u\n",(unsigned int)a+b);

printf("a +b = %u\n",(int)a+b);

//结果正常

unsigned int aa = 10;

int bb = -5;

printf("aa + bb = %u\n",aa+bb);//混合提升 将int类型的数据 转换为了unsigned int

printf("aa +bb = %u\n",(unsigned int)aa+bb);

printf("aa +bb = %u\n",(int)aa+bb);

return 0;

}

1. 程序：程序中包含2个区域：text/initial data
2. 进程：进程包含5个区域，分别是：text /initial data /uninitial data /heap /stack
3. 栈(stack)：先进(压栈)后出(出栈)
4. 调用的函数的参数也要压栈
5. ！是逻辑取反，只要数值不是0，在前面+！后都是0
6. 传递二维数组指针：假设要传递int[3][4]的指针，本质上要传递 int[4] \*p,但是编译无法通过，改为int \*p[4](但是这样写，就成为了指针数组，这个数组里四个元素都是int \* 型的),真正的数组指针int(\*p)[4]（指针的步长是一个数组），范围的话就是[3](也就是3)
7. 定义一个数组指针的变量： char (\*p)[10];-> typedef char (\*aP(别名))[10] aP a;//就定义了一个数组指针变量a
8. 数组指针的强制转化：int(\*)[4]
9. 指针的本质：有类型的地址
10. Int \* p : \* 声明这是一个指针同时也指定了大小 int 代表指针类型，决定寻址能力
11. & \* 可以实现一级指针(数组名)，到数组指针之间的切换。
12. 能通过二级指针，实现了对数据空间的间接访问 \*\*ppc ；能通过二级指针，改变一级指针的指向问题。
13. 能通过1级指针（\*）可以改变0级(基本类型)的内容
14. 通过2级指针，可以改变1级指针的指向问题
15. 通过3级指针，可以改变2级指针的指向问题
16. 二级指针的步长为4，其他类型的指针的寻址能力根据类型而定。
17. 一级指针作返回值输出：输入最小单元的大小以及行列，得到线性的内存空间，然后用数组指针去解释，从而在逻辑上是二维空间，类似于二维数组
18. 二级指针作返回值输出:输入行，得到指针数组，输入单元大小和列的大小，得到一维数组的大小。此时也能得到一个二维的逻辑空间。跟二维数组的本质，相去甚远。
19. 三级指针做参数输出：在二维指针作返值输出的基础上，将二维指针作参数输入。得到同二级指针作返值输出二维空间是一样的。
20. \*\* ：使人联想到一维数组 或者纯粹作为二级指针
21. Const 取代宏常量 宏的特点是替换 Const修饰的变量务必要初始化
22. Const修饰变量 去掉类型见本质

Const int \*p = &a; //表示指针指向的内容不可以改变

上面等价于int const \*p = &a; //p指向的内容不可以改变，可以改变的是指针本身的指向

int \* const p = &a;//表示修饰的指向不可以改变，但是指向的内容可以改变

const int \*const p = &a;//表示修饰的指向不可以改变，指向的内容也不可以改变

1. 函数名的本质：代表了一段可执行性代码的首地址
2. 函数类型的基本格式：void(\*p)()void
3. (\*(void(\*)())0)() <-> ((void(\*)())0)() 含义：返回值为void，参数为void，地址为0的函数调用 或者 一个0地址的返回值和参数值皆为void的一个指针函数的调用
4. Void (\*p)(void) = print p() <-> void (\*p)(void) = &print (\*p)()
5. Int arr[10] <-> int \*p +10 <-> p = arr ; int arr[10] <-> type arr <-> int(\*p)[10] = &arr
6. Int(\*p)[3][4] = &arr(int arr[3][4]) ：二维数组指针 合理性的过程:int(\*p)[3][4] -> int[4] (\*p)[3] -> type (\*p)[3]
7. 指针 改指向：就是改变指针中原来保存的谁的地址，换成另一个的地址 纯传指针值：可以改内容，主要是将原来保存的谁地址里的内容给改了