什么是“定义变量”？定义变量就是 **类型名 变量名**
例如：int x; int y = 1; void fun(int a[]); char *p; void fun(int &b); 分别定义了 x、y、a、p、b。
没有类型名就不是定义！【把这句话读三遍，再和下面提到“定义”的知识点结合起来看】

1、指针的基本概念

指针是一种数据，它存储的是内存地址。
指针分为不同类型（如 int*、char*），用来表示不同类型的数据（如 int、char）的地址。
指针变量本身的大小和它的指针类型无关。在 32 位系统中，指针变量的大小总是 4 字节。

如果 p 是指向变量 x（或对象 x）的指针，那么 p 里存储的数据是&x（即 x 的地址）。
如果 p 是指向数组 y 的首个元素的指针，那么 p 里存储的数据是 y（即数组名），**不是&y！**

不能用常量来表示变量的地址并给指针赋值，因为常量是要在编译前确定的，而变量的地址是在编译后才确定的。

2、“&”符号和“*”符号

地址运算符&后面不能跟常量或非赋值表达式，例如 &2、&(a+b)都是错误的。
C++中的&符号除了取地址外，还用在**引用的定义**中（如果不是定义，&就不表示引用），
例如 int &x = a; // 表示 x 是个引用。注意：这种用法和地址**无关**！
【不能看到&就只想到取地址！也不能看到&就只想到引用！要具体问题具体分析】

C++中的星号（*）有三种用法，第①种是表示乘号，第②、③两种都和指针有关。
第②种用法是在**定义**指针变量时，用星号（*）表示该变量是指针类型，例如：
int *p; 或 int *p = &x; 或 double *SavingAccount::p = NULL;
注意，使用指针变量**本身**时，**不加**星号（*）！例如：
p = &x; 或 p1 = p2; 或 p = p+1; 或 p = new double[20]; 或 delete p;
第③种用法是在使用指针变量指向的对象时，用星号（*）表示该对象，例如：
*p = 8; 或 x = *p; 或 *p1 = *p2; 或 cout << *p;

3、空指针和野指针

NULL 的值一般是 0，表示一个不能存储任何东西的地址。不能写成 Null。
值为 NULL 的指针称为空指针。

定义了一个指针变量后，它存储的是一个随机的地址。**不能**访问**随机**的地址！例如：
char *s; strcpy(s, “hello”); 是错的！（逻辑错误，编译器不检查）
值为随机的指针称为野指针。

4、指针与数组

通过指针操作数组元素有两种方式，无论哪种，首先都要让指针指向一个数组元素，例如：
int *p, a[10]; p = a; //相当于 p = &a[0]; 或者 p = &a[2]; // 相当于 p = a+2;

在此基础上，第①种是操作这个指针指向的元素，例如 *p *= 2;
第②种是基于这个指针计算其它的地址，再使用计算后的地址，例如 cout << *(p+2);

一定一定一定要掌握指针（包括数组名）的加减计算和赋值（包括传参）!
例如：形参是 int a[]，实参是 a+1。首先要明白这两个 a 相互独立，其次要明白形参 a 是一个指针，它通过传参被初始化为实参的值。

对于指针 p，p+1 的值可能**不等于** p 中存储的地址加 1，而与 p 的类型有关。
例如对于 char *、int *、double *的 p，p+1 的值分别等于 p 中存储的地址加 1、4、8。
同理，如果 int *指针 p 指向 int 数组 a 中的元素 a[3]，那么 p-a 的值为 3（而不是 12）。

不能用二维数组名给一级指针赋值。例如 int a[5][5], *p; p = a; 是错误的!
要让一级指针 p 获得该二维数组的起始地址，要写成 p = a[0]; 或 p = &a[0][0];

5、动态内存分配

使用 new 时，注意区分小括号和中括号。
p = int new(5); // 定义了单个动态变量并初始化为 5，让 p 指向该变量
p = int new[5]; // 定义了一个长度为 5 的动态数组，让 p 指向该数组的首个元素
如果在使用 new 的时候用了中括号，那么 delete 的时候也加上中括号，反之则不加。
如果 new 操作发现堆空间的剩余内存不足，那么会将指针赋值为 NULL。
【因为 NULL 的值是 0，而 0 就是 false，所以可以通过 if (p)来判断 new 操作是否成功】

用 new 创建动态数组时，数组大小可以用变量规定，但该变量的值**必须**是在运行到 new 语句时已经确定的。即在创建动态数组之前，要先给该变量赋值。

注意 delete p **不是**把指针 p 删掉，而是把 p 指向的堆空间释放，**p 本身没有任何变化**!

如果指向动态空间的指针变量消亡了或被修改了，且该指针变量原来的值没有保存在别的指针变量中，那么程序将无法访问也无法释放这片动态空间，造成内存泄漏。

6、指针与字符串

C++中的字符串常量有两种用法，第①种是给字符数组赋初值，例如：
char a[] = “good”; //字符串常量中的每个字符给对应的数组元素赋初值，数组大小为 5
注意**不能**写成 char a[5]; a = “good”; // 给数组名赋值是错误的
但可以写成 char a[5]; cin >> a; 以及 strcpy(a, “good”);
第②种是提供字符串常量的起始地址，例如：
const char *p = “good”; // 将指针 p 的初值设为字符串常量“good”的起始地址

关于 cstring 库的函数 strcpy 和 strcat，第一个参数必须表示一个字符数组的起始地址。
【因此第一个参数可以是字符数组的数组名，也可以是指向字符数组的指针。】
第二个参数必须表示一个字符串（也就是必须以‘\0’结尾）。

7、指针与函数

当函数的形参是指针时，实参应是相同类型的地址（变量的地址、同类指针或数组名）。

```
void myfun (int *p) { }

函数的调用：
int x;   myfun (&x); //  变量的地址
int *q = &x; myfun (q); //  同类指针，注意这里实参没有星号，是指针本身
int a[10]; myfun (a); //  数组名
```

当函数的返回值是指针时，返回地址对应的变量不能是局部变量（返回类型为引用时同理）。
【这不是语法要求，而是逻辑要求。因为局部变量会消亡，消亡的变量及其地址没有意义】
注意在函数中用 new 创建的动态变量不是局部变量。
【动态变量指的是在堆空间中的那个没有名字的变量；指针本身是局部变量】

8、引用

定义“指针的引用”时，先写*，后写&，例如：int *&q = p;
不能建立引用的指针或引用的数组。

实参为非指针类型的变量时，形参可以是同类型的变量（值传递）或引用（引用传递）。
【要根据传参是否调用拷贝构造函数、在被调函数中是否修改实参等来决定选择哪种形参】

函数的返回类型为引用时，可将函数调用表达式视作被调函数的返回值在主调函数中的别名，该函数调用表达式可以用作赋值表达式的左值。

9、二级指针和数组指针

如有定义 int **p, (*q)[5], a[5][5]; 那么 p = a; 是错误的，而 q = a; 是正确的。
【尽管二维数组名的“地位”相当于二级指针，但它的“列数”是常量，只能给“列数”为相同常量的数组指针赋值。】

注意 char **p; p = new char*[10]; strcpy(p[0], “Zero”); 是错误的，因为这段代码只给二级指针 p 分配了动态数组，却没有给一级指针 p[0]分配动态数组，所以 p[0]指向随机的内存空间，不能把字符串”Zero”复制到一个随机的地方。
类似地，char **p; *p = new char[10]; 也是错的，因为还没有给 p 指定对象（即给 p 赋值），所以不能用星号（解引用运算符）来访问它的对象*p。