


【C++语法】Lesson 3 讲义

【C++语法】Lesson 3 讲义

1. 代码模板
 - 1.1. 万能头文件
 - 1.2. define宏定义
 - 1.3. typedef类型定义
 - 1.4. const关键字
 - ※ 全局变量与局部变量，堆空间与栈空间
 - ※ 变量作用域问题
 - 1.5. scanf快读
2. 函数
 - 2.1. 函数的定义
 - 2.2. 函数的调用
 - 2.3. 值传递问题
 - 2.3.1. 形参改变，不影响实参
 - 2.3.2. &引用，形参改变，影响实参
 - 2.4. 函数常见形式
 - 2.5. 函数的声明
 - 2.7. 函数分文件编写
 - 2.6. 函数的递归调用
2. 快速掌握调试技巧
3. 指针
 - 3.1. 指针变量的定义和使用
 - 3.2. 指针所占内存空间
 - 3.3. 空指针和野指针
 - 3.3.1. 空指针
 - 3.3.2. 野指针
 - 3.4. 常量指针与指针常量
 - 3.5. 指针和数组
 - 3.6. 指针和函数
4. 结构体
 - 4.1. 结构体的定义和使用
 - 4.2. 结构体数组
 - 4.3. 结构体指针
 - 4.4. 结构体嵌套
5. 类与模板

本文档为程序设计算法协会C++语法课培训第三节课的讲义，制
作人：漂至 

1. 代码模板

- 在之前的学习中，我们已经掌握了最基础的模板

```

1 #include<iostream>
2 using namespace std;
3
4 int main() {
5
6     return 0;
7 }

```

- 但是在算法竞赛中我们可以为自己书写一个更为方便、高阶的代码模板，以下是我比较常用的模板，分享给大家，然后介绍一系列要点

```

1 #include<bits/stdc++.h>
2 #define x first
3 #define y second
4
5 using namespace std;
6
7 typedef long long LL;
8 typedef pair<int,int> PII;
9
10 /*
11     解题思路：
12
13 */
14
15 const int N=1e5+5;
16
17 int n,a[N];
18
19
20 int main() {
21     cin>>n;
22     for(int i=1;i<=n;i++) {
23         scanf("%d",&a[i]);
24     }

```

```
25
26     return 0;
27 }
28
29 /*
30  输入样例:
31
32  输出样例:
33
34  */
```

1.1. 万能头文件

- 在写代码的时候，我们会引入非常多的头文件，比如 `iostream`、`cstring`、`climits`，万能头文件 `bits/stdc++.h` 包含了我们常用的一系列头文件，只需要引用它就可以避免书写很多头文件（头文件很多很容易记错），同时节约写代码的时间
- PS：头文件变多意味着编译时间也会变多，但算法竞赛中的时间限制不包含编译时间

```
1 #include<bits/stdc++.h>
```

1.2. define 宏定义

- 宏定义是在C/C++编程中用于创建标识符与代码片段之间的映射。它通过预处理器（preprocessor）在编译前进行文本替换

```
1 #define 宏名 替换文本
2 #define PI 3.14159
```

- 在模板代码中，用x去替换first，用y去替换second，这是因为在后续我们学习STL库的时候能够更方便地访问pair、map等类型中的元素

```
1 #define x first
2 #define y second
```

- 举个例子提前学习一下

```
1 #include<bits/stdc++.h>
2 #define x first
3 #define y second
4
5 using namespace std;
6
7 int main() {
8     // pair n.一对
9     pair<int,int> p(3,4);
10    cout<<p.first<<' '<<p.second<<'\n';
11    cout<<p.x<<' '<<p.y<<'\n';
12    return 0;
13 }
```

1.3. typedef类型定义

```
1 typedef long long LL;
2 typedef pair<int,int> PII;
```

- typedef 用于定义类型别名，这里将long long取了个别名为LL，上个案例中用到的pair<int,int>取别名为PII，现在我们就可以用别名来代替原类型名

```
1 #include<bits/stdc++.h>
2 #define x first
```

```

3 #define y second
4
5 using namespace std;
6
7 typedef long long LL;
8 typedef pair<int,int> PLL;
9
10 int main() {
11     // pair n.一对
12     PLL p(3,4);
13     cout<<p.first<<' '<<p.second<<'\n';
14     cout<<p.x<<' '<<p.y<<'\n';
15
16     LL a=3;
17     cout<<sizeof(a)<<'\n'; // 8字节
18     return 0;
19 }

```

1.4. const关键字

```

1 const int N=1e5+5;

```

- 这里定义了一个常量 `N`，表示可以使用的数组的最大大小（也就是我们常说的问题规模）
- `1e5` 表示 `100000`，加 `5` 是为了保证有足够的空间（数组大小为 `100000`，则下标为 `[0,99999]`，一般来说我们从下标 `1` 开始存储，为存储最后一个元素，下标应开到 `100000`，故数组大小应为 `100001`，所以在后面任意加一个常量，一般加个 `5` 加个 `10` 都是可以的，避免下标越界

※ 全局变量与局部变量，堆空间与栈空间

```
1 | int n,a[N]; // 定义在main函数外，为全局变量
```

- `main`函数之前被定义的变量称为全局变量，全局变量使用堆空间，可容纳的数据范围更大，作用域为整个文件
- `main`函数以内被定义的变量称为局部变量，局部变量使用栈空间，可容纳的数据范围更小（所以大数组我们都开在全局变量），作用域取决于大括号`{}`

※ 变量作用域问题

- 同一作用域中不得出现相同变量名的变量！

```
1 | int main() {  
2 |     int k;  
3 |     k的作用域在本括号内;  
4 |     for(int i=1;i<=n;i++) {  
5 |         i的作用域在本括号内;  
6 |         for(int j=1;j<=n;j++) {  
7 |             j的作用域在本括号内;  
8 |         }  
9 |     }  
10| }
```

1.5. scanf快读

- 很多算法题目都需要依赖数组实现，一般会先让我们输入数组中元素的个数，再输入每个元素的具体值，所以先输入`n`，再通过`scanf`输入数组，称为快读

```
1 int main() {
2     cin>>n;
3     for(int i=1;i<=n;i++) {
4         scanf("%d",&a[i]);
5     }
6
7     return 0;
8 }
```

2. 函数

本小节讲义参考黑马程序员C++入门讲义

- 函数的作用是将一段经常使用的代码封装起来，减少重复代码，提高代码的复用性
- 一个较大的程序，一般分为若干个程序块，每个模块实现特定的功能（分层解耦思想）

2.1. 函数的定义

- 一个函数一般由以下5要素构成：
 1. 返回值类型：一个函数可以返回一个值，确定这个值的类型
 2. 函数名：给函数起个名字
 3. 参数表列：使用该函数时，需要传入的数据
 4. 函数体语句：花括号内的代码，函数内需要执行的语句
 5. return 表达式：和返回值类型挂钩，函数执行完后，返回相应的数据

```
1 返回值类型 函数名 （参数列表）
2  {
3     函数体语句；
4     return表达式；
5 }
```

- 比如定义一个加法函数，实现两个数相加

```
1 int add(int a,int b) {  
2     int sum=a+b;  
3     return a+b;  
4 }
```

2.2. 函数的调用

- 即使用定义好的函数

```
1 #include<bits/stdc++.h>  
2 using namespace std;  
3  
4 // 既然是函数，那肯定与main函数同级咯  
5 // 在这里声明函数  
6 int add(int a,int b) {  
7     int sum=a+b;  
8     return a+b;  
9 }  
10  
11 int main() {  
12     // 声明了才可以调用，就像变量一样！  
13     cout<<add(3,5)<<'\n';  
14  
15     int a,b;  
16     cin>>a>>b;  
17     cout<<add(a,b)<<'\n';  
18  
19     int c,d;  
20     cin>>c>>d;  
21     cout<<add(c,d)<<'\n';  
22 }
```


- 我们发现函数的参数名叫做 `a,b`，我们在 `main` 函数中定义变量时取名也叫 `a,b` 为什么没有冲突呢？
- 这是因为变量名中的 `a,b` 叫做形参，没有实际意义，实际定义的变量名可以叫 `a,b` 也可以叫其他名字，这是实际传递给函数的参数，我们称其为实参

2.3. 值传递问题

2.3.1. 形参改变，不影响实参

- 输入 `3 5`，可以看到，在调用函数前后，`a=3`

```
1 // 让a加上x
2 int add(int a,int x) {
3     a+=x;
4     return a;
5 }
6
7 int main() {
8     int a,b;
9     cin>>a>>b;
10    cout<<a<<'\n'; // 原来的值
11    cout<<add(a,b)<<'\n';
12    cout<<a<<'\n'; // 现在的值
13    return 0;
14 }
```

2.3.2. &引用，形参改变，影响实参

- 输入 `3 5`，调用前 `a=3`，调用后 `a=8`

```
1 // 让a加上x
2 int add(int &a,int x) {
```

```

3     a+=x;
4     return a;
5 }
6
7 int main() {
8     int a,b;
9     cin>>a>>b;
10    cout<<a<<'\\n'; // 原来的值
11    cout<<add(a,b)<<'\\n';
12    cout<<a<<'\\n'; // 现在的值
13    return 0;
14 }

```

2.4. 函数常见形式

- 常见的形式有四种：① 无参无返、② 有参无返、③ 无参有返、④ 有参有返
- 这里用到了我们之前讲基本数据类型中的空类型 `void`，`void` 可以修饰函数类型，表示空函数，即没有返回值

```

1 // 函数常见样式
2 // 1、 无参无返
3 void test01()
4 {
5     //void a = 10; //无类型不可以创建变量,原因无法分配
    内存
6     cout << "this is test01" << endl;
7     //test01(); 函数调用
8 }
9
10 // 2、有参无返
11 void test02(int a)
12 {
13     cout << "this is test02" << endl;

```

```

14     cout << "a = " << a << endl;
15 }
16
17 // 3、无参有返
18 int test03()
19 {
20     cout << "this is test03 " << endl;
21     return 10;
22 }
23
24 // 4、有参有返
25 int test04(int a, int b)
26 {
27     cout << "this is test04 " << endl;
28     int sum = a + b;
29     return sum;
30 }

```

2.5. 函数的声明

- 函数和变量一样，在使用前必须先声明，函数可以声明多次，但函数的定义只能有一次

```

1 // 声明可以多次，定义只能一次
2 // 声明
3 int max(int a, int b);
4 int max(int a, int b);
5 // 定义
6 int max(int a, int b)
7 {
8     return a > b ? a : b;
9 }
10
11 int main() {

```

```
12     int a = 100;
13     int b = 200;
14     cout << max(a, b) << endl;
15     return 0;
16 }
```

2.7. 函数分文件编写

- 我们已经知道函数本身的设计目的就是用于实现特定的功能，那我们可以将代码分文件编写来让代码的结构显得更加清晰
- 函数分文件编写一般有4个步骤：
 1. 创建后缀名为 `.h` 的头文件
 2. 创建后缀名为 `.cpp` 的源文件
 3. 在头文件中写函数的声明
 4. 在源文件中写函数的定义
- 也就是说，函数的声明写在 `.h` 源文件中，函数的实现写在 `.cpp` 源文件中

```
1 // swap.h文件
2 #include<iostream>
3 using namespace std;
4
5 //实现两个数字交换的函数声明
6 void swap(int a, int b);
```

```

1 // swap.cpp文件
2 #include "swap.h"
3
4 void swap(int a, int b)
5 {
6     int temp = a;
7     a = b;
8     b = temp;
9
10    cout << "a = " << a << endl;
11    cout << "b = " << b << endl;
12 }

```

- 就像我们引入的那些头文件一样，本质上他们也通过这样的形式实现了一系列功能，在 `main` 函数文件中，我们就可以通过 `#include` 指令来包含我们自己写的头文件 `swap.h`
- 随后就可以使用 `swap.c` 中的 `swap` 函数

2.6. 函数的递归调用

- 递归：函数直接或间接调用自身的过程，用于将复杂的问题分解成更简单的子问题，直到子问题简单到可以直接求解为止，随后从下往上合并子问题的答案，得到最终结果，是**分治算法的基础思想**
- 一个典型的递归函数由基本情况（可以直接求解）和递归调用构成

```

1 // 求阶乘的递归函数
2 int factorial(int n) {
3     if (n == 0) { // 基本情况
4         return 1;
5     }
6     return n * factorial(n - 1); // 递归调用

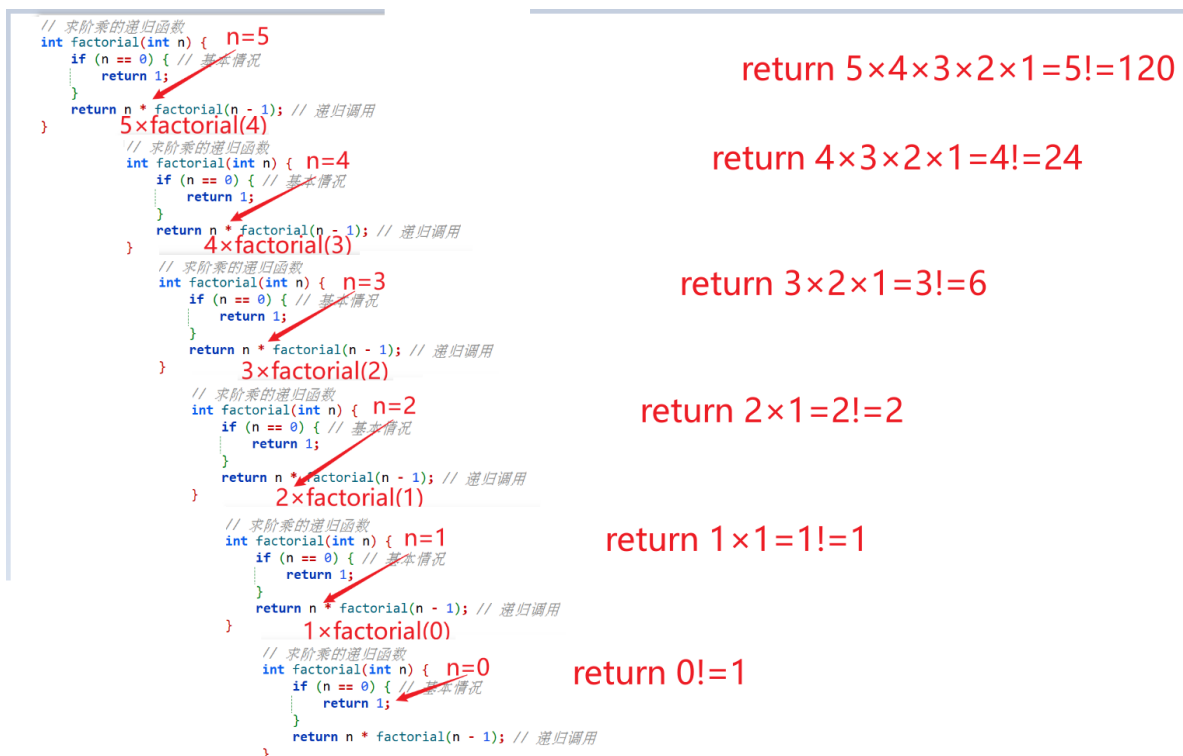
```

```

7 }
8
9 int main() {
10     int num;
11     cout << "请输入一个非负整数:>";
12     cin >> num;
13     cout << num << " 的阶乘是: " << factorial(num)
14     << endl;
15     return 0;
16 }

```

- 如上述案例中计算 5 的阶乘即 `factorial(5)`，实际函数调用流程如下：



2. 快速掌握调试技巧

- 调试是用来帮助我们找出代码中的错误Debug后重新提交，保证AC效率，在比赛中不输人的常用技巧

- 不同IDE关于调试的操作可能不一样，这里我们同样以小熊猫C++为例，调试快捷键为F5
- 在调试之前，需要把编译器版本改成 Debug



- 案例：首先输入 **n**，再向数组输入 **n** 个数，进行 **m** 次修改操作，每次修改的格式为 **a b**，表示让第 **a** 个元素加上 **b**，随后输出这 **n** 个数中的最大值和最小值之差

```

1  #include<bits/stdc++.h>
2  #define x first
3  #define y second
4
5  using namespace std;
6
7  typedef long long LL;
8  typedef pair<int,int> PII;
9
10 /*
11     解题思路： 首先快读n个数
12
13     */
14
15 const int N=10; // 数组范围限制小方便调试，但是提交代码
                  // 的时候注意把问题规模该回去哦！
16
17 int n,a[N];
18 int m;
19
20 int add(int x,int y) {

```

```

21     a[x]+=y; // a是全局变量，所以可以直接用，如果a是局
    部变量，就要作为参数传递进来
22 }
23
24 int main() {
25     // 输入n个数
26     cin>>n;
27     for(int i=1;i<=n;i++) {
28         scanf("%d",&a[i]);
29     }
30     // 输入m组操作
31     cin>>m;
32     for(int i=1;i<=m;i++) {
33         int x,y;
34         cin>>x>>y; // 把第x个元素的值改为y
35         add(x,y);
36     }
37     // 找出最大值和最小值
38     int mmax=INT_MIN, mmin=INT_MAX; // INT_MIN:
    int型的最小值，找最大赋极小，找最小赋极大
39     for(int i=1;i<=n;i++) {
40         if(a[i]>mmax) mmax=a[i];
41         if(a[i]<mmin) mmin=a[i];
42     }
43     cout<<mmax-mmin<<"\n";
44     return 0;
45 }
46
47 /*
48 输入样例：
49 7
50 1 3 2 5 9 7 4
51 3
52 1 3
53 2 -1
54 6 6

```


55 输出样例:

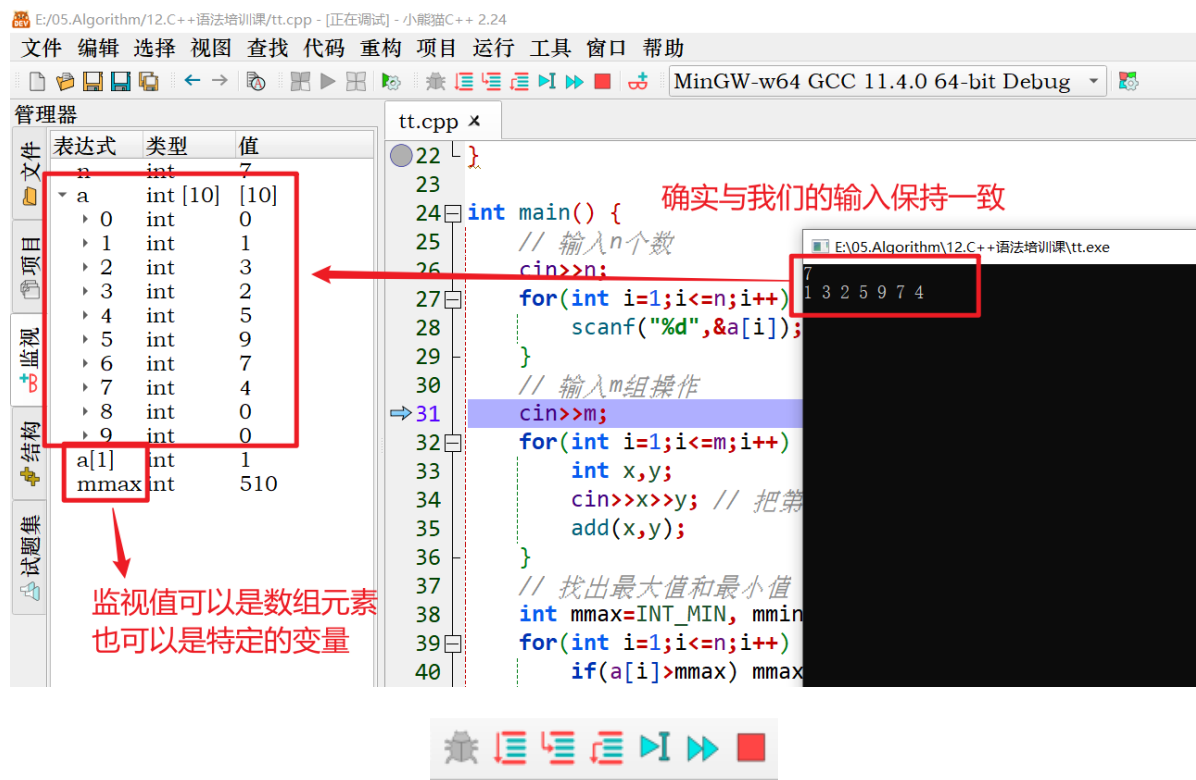
56 11

57 */

- 我们想通过调试看一下数组中的值究竟是如何变化的:
- 首先我们要先在代码行左侧点击一下打一个断点(断点是调试的入口),断点以前的地方通常是我们自认为已经没有问题、没必要检查的地方,断点之后的地方是我们认为出错的地方,随后按下F5进入调试

```
tt.cpp x
22 }
23
24 int main() {
25     // 输入n个数
26     cin>>n;
27     for(int i=1;i<=n;i++) {
28         scanf("%d",&a[i]);
29     }
30     // 输入m组操作
31     cin>>m;
32     for(int i=1;i<=m;i++) {
33         int x,y;
34         cin>>x>>y; // 把第x个元素的值改为y
35         add(x,y);
36     }
37     // 找出最大值和最小值
38     int mmax=INT_MIN, mmin=INT_MAX; // INT_MIN: int型的最小值, 找最大赋极
39     for(int i=1;i<=n;i++) {
40         if(a[i]>mmax) mmax=a[i];
41         if(a[i]<mmin) mmin=a[i];
42     }
```

- 然后按照顺序的结构依次输入需要的元素, 在左侧的监视窗口中
添加我们需要监测其值变化情况的变量



- 在调试的时候，程序一步一步进行，通过上方状态栏完成（当进入调试模式时图标才会亮）【该部分请结合我的演示】：
 - 单步跳过(Step Over)：一行程序算一步，执行完当前行后暂停（执行一步，跳过函数）
 - 单步进入(Step Into)：如果当前行不包含函数调用，则一行程序算一步；如果这行程序中包含对函数的调用，会在进入函数后暂停；如果找不到该函数的符号信息，则在执行完该函数后暂停（执行一步，进入函数）
 - 单步跳出(Step Out)：退出当前函数后暂停（退出函数）
 - 执行到光标处：顾名思义，直接跳转到你光标所在的地方
 - 继续执行：执行到下一个断点

3. 指针

- 指针也是一种变量，用于存储一个变量的内存地址，通过指针，可以间接访问和操作内存中的数据
 - 内存编号是从0开始记录的，一般用十六进制数字表示
 - 可以利用指针变量保存地址

3.1. 指针变量的定义和使用

- 指针变量定义语法： `数据类型 * 变量名;`

```
1  int main() {
2
3      // 1、指针的定义
4      int a = 10; //定义整型变量a
5      // 指针定义语法： 数据类型 * 变量名 ;
6      int * p;
7
8      // 指针变量赋值
9      // 通过 & 符号 获取变量的地址
10     p = &a; // 指针指向变量a的地址(利用指针可以记录地
        址)，在讲scanf的时候我们提过&是取地址符
11     cout << &a << endl; // 打印数据a的地址
12     cout << p << endl; // 打印指针变量p
13
14     // 2. 指针的使用
15     // 通过*操作指针变量指向的内存，此为指针解引用操作
16     // 对指针变量解引用，可以操作指针指向的内存
17     cout << "*p = " << *p << endl;
18
19     return 0;
20 }
```

- 指针变量和普通变量的区别：
 - 普通变量存放的是数据，指针变量存放的是地址
 - 指针变量可以通过"*"操作符，操作指针变量指向的内存空间，这个过程称为解引用

3.2. 指针所占内存空间

- 指针也是种数据类型，那么这种数据类型占用多少内存空间？
- 运行下面这段代码，我们会发现：所有指针类型在32位操作系统下是4个字节，这是因为32位系统中的内存地址是32位的，同理，在64位操作系统下是8个字节

```
1  int main() {  
2      int a = 10;  
3  
4      int * p;  
5      p = &a; //指针指向数据a的地址  
6  
7      cout << *p << endl; /* 解引用  
8      cout << sizeof(p) << endl;  
9      cout << sizeof(char *) << endl;  
10     cout << sizeof(float *) << endl;  
11     cout << sizeof(double *) << endl;  
12  
13     return 0;  
14 }
```

3.3. 空指针和野指针

3.3.1. 空指针

- 空指针：指针变量指向内存中编号为0的空间，用于初始化指针变量，PS：空指针指向的内存是不可以访问的！

```

1  int main() {
2      // 指针变量p指向内存地址编号为0的空间
3      int * p = NULL;
4
5      // 访问空指针报错
6      // 内存编号0~255为系统占用内存，不允许用户访问
7      cout << *p << endl;
8
9      return 0;
10 }

```

3.3.2. 野指针

- **野指针：**① 指针变量没有被初始化（操作系统会随机分配到一块内存），② 指向非法的内存空间；这两种情况的本质都是指针指向的地址没有被申请，无法使用
- 所以指针一定一定要初始化，保证安全性

```

1  int main() {
2      // 指针变量p指向内存地址编号为0x1100的空间
3      int * p = (int *)0x1100; // 0x1100是int型变量，
    需要强转，这个地址没有被申请，无权操控
4      // 访问野指针报错
5      cout << *p << endl;
6
7      // 指针未初始化
8      int *pp;
9      cout<< *pp <<endl;
10
11     return 0;
12 }

```

3.4. 常量指针与指针常量

- `const` 可以修饰变量变为常量，同样`const`也可以作用于指针
 - `const` 修饰指针，如 `const int * p`，被称作常量指针，指针指向可以改，但无法间接修改指针所指向数据的值
 - `const` 修饰常量，如 `int const *p`，被称作指针常量，指针指向不可以改，但可以间接修改指针所指向数据的值
- PS：看`const`右侧紧跟着的是指针还是常量，是指针就是常量指针，是常量就是指针常量

```
1  int main() {
2      int a = 10;
3      int b = 10;
4
5      // const修饰的是指针，指针指向可以改，指针指向的值不
      可以更改
6      const int * p1 = &a;
7      p1 = &b; // 正确
8      /*p1 = 100; // 报错
9
10     // const修饰的是常量，指针指向不可以改，指针指向的值
        可以更改
11     int * const p2 = &a;
12     //p2 = &b; //错误
13     *p2 = 100; //正确
14
15     // const既修饰指针又修饰常量
16     const int * const p3 = &a;
17     // p3 = &b; //错误
18     // *p3 = 100; //错误
19
20     return 0;
21 }
```

3.5. 指针和数组

- 作用：利用指针访问数组中的元素

```
1  int main() {
2      int arr[] = { 1,2,3,4,5,6,7,8,9,10 };
3      int * p = arr; //指向数组的指针
4      cout << "第一个元素: " << arr[0] << endl;
5      cout << "指针访问第一个元素: " << *p << endl;
6      for (int i = 0; i < 10; i++) {
7          //利用指针遍历数组
8          cout << *p << endl;
9          p++;
10     }
11
12     return 0;
13 }
```

3.6. 指针和函数

- 作用：利用指针作函数参数，可以修改实参的值，和&引用类似，都是直接修改地址
- 如果不想修改实参，就用值传递，如果想修改实参，就用地址传递
- 注意地址传递和引用的区别！

```
1  // 值传递
2  void swap1(int a ,int b)
3  {
4      int temp = a;
5      a = b;
6      b = temp;
7  }
8  // 地址传递
```

```

9 void swap2(int * p1, int *p2)
10 {
11     int temp = *p1;
12     *p1 = *p2;
13     *p2 = temp;
14 }
15 // 之前讲过，也可以这样写
16 void swap3(int &p1, int &p2) {
17     int tmp=p1;
18     p1=p2;
19     p2=tmp;
20 }
21 int main() {
22     int a = 10;
23     int b = 20;
24     swap1(a, b); // 值传递不会改变实参
25     swap2(&a, &b); // 地址传递会改变实参
26     swap3(a,b); // 引用
27     cout << "a = " << a << endl;
28     cout << "b = " << b << endl;
29     return 0;
30 }

```

4. 结构体

该部分参考黑马程序员C++讲义

- 结构体是一种用户自定义的数据类型，允许用户存储不同的数据类型

4.1. 结构体的定义和使用

- 语法： `struct 结构体名 { 结构体成员列表 };`
 - 定义结构体时的关键字是struct，不可省略

- 创建结构体变量时，关键字struct可以省略
- 结构体变量利用操作符"." 访问成员

```
1 struct 结构体名 变量名;  
2  
3 struct 结构体名 变量名 = { 成员1, 成员2值}  
4  
5 struct 结构体名 {  
6     成员1,  
7     成员2,  
8     ...  
9 }变量名;
```

```
1 //结构体定义  
2 struct student {  
3     //成员列表  
4     string name; //姓名  
5     int age; //年龄  
6     int score; //分数  
7 }stu3; //结构体变量创建方式3  
8  
9 int main() {  
10     //结构体变量创建方式1  
11     struct student stu1; //struct 关键字可以省略  
12     stu1.name = "张三";  
13     stu1.age = 18;  
14     stu1.score = 100;  
15     cout << "姓名: " << stu1.name << " 年龄: " <<  
16     stu1.age << " 分数: " << stu1.score << endl;  
17  
18     //结构体变量创建方式2  
19     struct student stu2 = { "李四", 19, 60 };  
20     cout << "姓名: " << stu2.name << " 年龄: " <<  
21     stu2.age << " 分数: " << stu2.score << endl;  
  
    stu3.name = "王五";
```

```

22     stu3.age = 18;
23     stu3.score = 80;
24     cout << "姓名: " << stu3.name << " 年龄: " <<
stu3.age << " 分数: " << stu3.score << endl;
25     return 0;
26 }

```

4.2. 结构体数组

- 结构体是我们自定义的数据类型，同样可以开数组
- 语法: `struct 结构体名 数组名[元素个数] = { {}, {}, ..., {} }`

```

1  struct student {
2      //成员列表
3      string name; //姓名
4      int age; //年龄
5      int score; //分数
6  };
7
8  int main() {
9      // 结构体数组
10     struct student arr[3]=
11     {
12         {"张三",18,80 },
13         {"李四",19,60 },
14         {"王五",20,70 }
15     };
16     for (int i = 0; i < 3; i++)
17     {
18         cout << "姓名: " << arr[i].name << " 年
龄: " << arr[i].age << " 分数: " << arr[i].score
<< endl;
19     }

```

```
20 |         return 0;
21 |     }
```

- 同样的，我们说数组可以开成全局变量，所以第三种定义方式其实更常用，我们再举个带输入输出的例子

```
1  struct student {
2      //成员列表
3      string name; //姓名
4      int age;      //年龄
5      int score;    //分数
6  }stu[3];
7
8  int main() {
9      // 输入
10     for (int i = 0; i < 3; i++)
11     {
12         cout<<"请输入第"<< i+1 << "个学生的姓名,年龄,
13         分数:>";
14
15         cin>>arr[i].name>>arr[i].age>>arr[i].score;
16     }
17     // 输出
18     for (int i = 0; i < 3; i++)
19     {
20         cout << "姓名: " << arr[i].name << " 年
21         龄: " << arr[i].age << " 分数: " << arr[i].score
22         << endl;
23     }
24     return 0;
25 }
```

4.3. 结构体指针

- 可以通过指针访问结构体中的成员，利用操作符 `->` 可以通过结构体指针访问结构体属性
 - 结构体指针访问属性使用： `->`
 - 结构体访问属性使用： `.`

```
1 //结构体定义
2 struct student
3 {
4     //成员列表
5     string name; //姓名
6     int age;      //年龄
7     int score;    //分数
8 }stu,*p;
9
10
11 int main() {
12     stu = { "张三",18,100 };
13     p = &stu;
14     p->score = 80; //指针通过 -> 操作符可以访问成员
15     cout << "姓名: " << p->name << " 年龄: " << p-
16     >age << " 分数: " << p->score << endl;
17     return 0;
18 }
```

4.4. 结构体嵌套

- 结构体中的成员可以是另一个结构体

```
1 // 学生结构体定义
2 struct student
3 {
4     //成员列表
```

```
5     string name;    // 姓名
6     int age;        // 年龄
7     int score;      // 分数
8 };
9
10 // 教师结构体定义
11 struct teacher
12 {
13     // 成员列表
14     int id; // 职工编号
15     string name; // 教师姓名
16     int age; // 教师年龄
17     struct student stu; // 子结构体 学生
18 }t1;
19
20
21 int main() {
22     struct teacher t1;
23     t1.id = 10000;
24     t1.name = "老王";
25     t1.age = 40;
26
27     t1.stu.name = "张三";
28     t1.stu.age = 18;
29     t1.stu.score = 100;
30
31     cout << "教师 职工编号: " << t1.id << " 姓名: "
32     << t1.name << " 年龄: " << t1.age << endl;
33     cout << "辅导学员 姓名: " << t1.stu.name << "
34     年龄: " << t1.stu.age << " 考试分数: " <<
35     t1.stu.score << endl;
36
37     return 0;
38 }
```

5. 类与模板

- 如果时间够就后续补充讲解