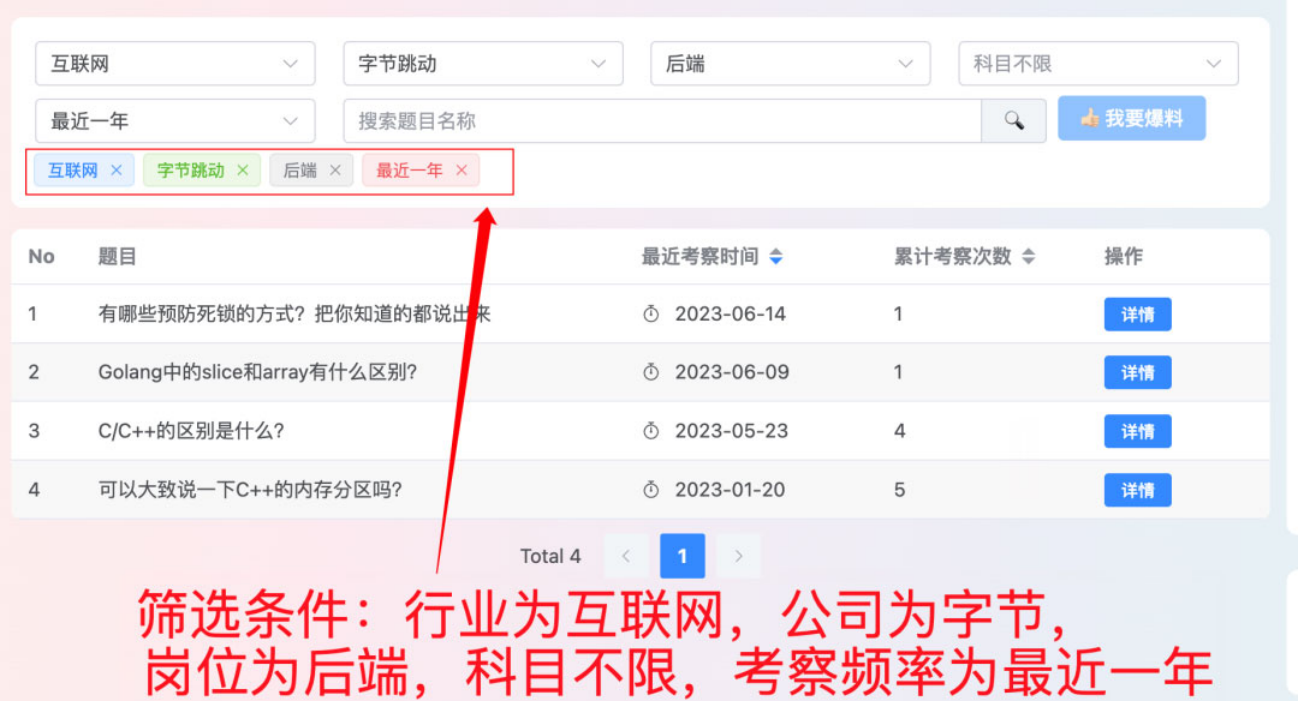
C++之基础语法

原创 C++ CPP 社招 校招 阿秀

这是四则或许对你有些许帮助的信息:

1、👉23年5月份我从字节跳动离职跳槽到某外企期间，为**方便自己找工作，增加上岸几率**，我自己从0开发了一个**大厂面试真题解析网站**，包括两个前端和一个后端。能够定向查看某些公司的某些岗位面试真题，比如我想查一下行业为互联网，公司为字节跳动，考察岗位为后端，考察时间为最近一年之类的面试题有哪些?



衷心希望自己开发的这个网站能够帮到更多的人，自己能够以技术服务于大家!

网站地址：InterviewGuide大厂面试真题解析网站。点此可以查看该网站的视频介绍：B站视频讲解 如果可以的话求个B站三连，感谢!

2、😍 免费分享阿秀个人学习计算机以来收集到的免费学习资源，点此白嫖；也记录一下自己以前买过的不错的计算机书籍、网络专栏和垃圾付费专栏。

> 经遇到过了。
>
> 4、🔥 欢迎准备计算机校招的小伙伴加入我的学习圈子，一个人踽踽独行不如一群人报团取暖，圈子里沉淀了很多过去21/22/23届学长学姐的经验和总结，好好跟着走下去的，最后基本都可以拿到不错的offer！此外，每周都会进行精华总结和分享！如果你需要《阿秀的学习笔记》网站中📚校招八股文相关知识点的PDF版本的话，可以点此下载 。

1、 在main执行之前和之后执行的代码可能是什么?

main函数执行之前，主要就是初始化系统相关资源：

- 设置栈指针
- 初始化静态 `static` 变量和 `global` 全局变量，即 `.data` 段的内容
- 将未初始化部分的全局变量赋初值：数值型 `short` ， `int` ， `long` 等为 `0` ，`bool` 为 `FALSE` ，指针为 `NULL` 等等，即 `.bss` 段的内容
- 全局对象初始化，在 `main` 之前调用构造函数，这是可能会执行前的一些代码
- 将main函数的参数 `argc` ， `argv` 等传递给 `main` 函数，然后才真正运行 `main` 函数
- `__attribute__((constructor))`

main函数执行之后：

- 全局对象的析构函数会在main函数之后执行；
- 可以用 `atexit` 注册一个函数，它会在main 之后执行;
- `__attribute__((destructor))`

> update1:https://github.com/forthespada/InterviewGuide/issues/2 ,由 `stanleyguo0207` 提出 - 2021.03.22

2、结构体内存对齐问题?

齐。）

c++11以后引入两个关键字 alignas 与 alignof 。其中 `alignof` 可以计算出类型的对齐方式， `alignas` 可以指定结构体的对齐方式。

但是 `alignas` 在某些情况下是不能使用的，具体见下面的例子:

```
// alignas 生效的情况

struct Info {
  uint8_t a;
  uint16_t b;
  uint8_t c;
};

std::cout << sizeof(Info) << std::endl;   // 6  2 + 2 + 2
std::cout << alignof(Info) << std::endl;  // 2

struct alignas(4) Info2 {
  uint8_t a;
  uint16_t b;
  uint8_t c;
};

std::cout << sizeof(Info2) << std::endl;   // 8  4 + 4
std::cout << alignof(Info2) << std::endl;  // 4
```

`alignas` 将内存对齐调整为4个字节。所以 `sizeof(Info2)` 的值变为了8。

```
// alignas 失效的情况

struct Info {
  uint8_t a;
  uint32_t b;
  uint8_t c;
```

```cpp
std::cout << sizeof(Info) << std::endl;   // 12  4 + 4 + 4
std::cout << alignof(Info) << std::endl;  // 4

struct alignas(2) Info2 {
  uint8_t a;
  uint32_t b;
  uint8_t c;
};

std::cout << sizeof(Info2) << std::endl;   // 12  4 + 4 + 4
std::cout << alignof(Info2) << std::endl;  // 4
```

若 `alignas` 小于自然对齐的最小单位，则被忽略。

- 如果想使用单字节对齐的方式，使用 `alignas` 是无效的。应该使用 `#pragma pack(push,1)` 或者使用 `__attribute__((packed))` 。

```cpp
#if defined(__GNUC__) || defined(__GNUG__)
  #define ONEBYTE_ALIGN __attribute__((packed))
#elif defined(_MSC_VER)
  #define ONEBYTE_ALIGN
  #pragma pack(push,1)
#endif

struct Info {
  uint8_t a;
  uint32_t b;
  uint8_t c;
} ONEBYTE_ALIGN;

#if defined(__GNUC__) || defined(__GNUG__)
  #undef ONEBYTE_ALIGN
#elif defined(_MSC_VER)
  #pragma pack(pop)
  #undef ONEBYTE_ALIGN
```

```
std::cout << sizeof(Info) << std::endl;   // 6 1 + 4 + 1
std::cout << alignof(Info) << std::endl;  // 1
```

- 确定结构体中每个元素大小可以通过下面这种方法:

```
#if defined(__GNUC__) || defined(__GNUG__)
  #define ONEBYTE_ALIGN __attribute__((packed))
#elif defined(_MSC_VER)
  #define ONEBYTE_ALIGN
  #pragma pack(push,1)
#endif

/**
* 0 1   3     6   8 9            15
* +-+---+-----+---+-+-------------+
* | |   |     |   | |             |
* |a| b |  c  | d |e|     pad     |
* | |   |     |   | |             |
* +-+---+-----+---+-+-------------+
*/
struct Info {
  uint16_t a : 1;
  uint16_t b : 2;
  uint16_t c : 3;
  uint16_t d : 2;
  uint16_t e : 1;
  uint16_t pad : 7;
} ONEBYTE_ALIGN;

#if defined(__GNUC__) || defined(__GNUG__)
  #undef ONEBYTE_ALIGN
#elif defined(_MSC_VER)
  #pragma pack(pop)
  #undef ONEBYTE_ALIGN
#endif
```

```
32      std::cout << alignof(Info) << std::endl;  // 1
33
```

这种处理方式是 `alignas` 处理不了的。

> update1:https://github.com/forthespada/InterviewGuide/issues/2 ,由 `stanleyguo0207` 提出 - 2021.03.22

3、指针和引用的区别

- 指针是一个变量，存储的是一个地址，引用跟原来的变量实质上是同一个东西，是原变量的别名
- 指针可以有多级，引用只有一级
- 指针可以为空，引用不能为NULL且在定义时必须初始化
- 指针在初始化后可以改变指向，而引用在初始化之后不可再改变
- sizeof指针得到的是本指针的大小，sizeof引用得到的是引用所指向变量的大小
- 当把指针作为参数进行传递时，也是将实参的一个拷贝传递给形参，两者指向的地址相同，但不是同一个变量，在函数中改变这个变量的指向不影响实参，而引用却可以。
- 引用本质是一个指针，同样会占4字节内存；指针是具体变量，需要占用存储空间（，具体情况还要具体分析）。
- 引用在声明时必须初始化为另一变量，一旦出现必须为typename refname &varname形式；指针声明和定义可以分开，可以先只声明指针变量而不初始化，等用到时再指向具体变量。
- 引用一旦初始化之后就不可以再改变（变量可以被引用为多次，但引用只能作为一个变量引用）；指针变量可以重新指向别的变量。
- 不存在指向空值的引用，必须有具体实体；但是存在指向空值的指针。

参考代码：

```
{
    int a=1;
    p=&a;
    cout<<p<<" "<<*p<<endl;
}

int main(void)
{
    int *p=NULL;
    test(p);
    if(p==NULL)
    cout<<"指针p为NULL"<<endl;
    return 0;
}
//运行结果为:
//0x22ff44 1
//指针p为NULL


void testPTR(int* p) {
    int a = 12;
    p = &a;

}

void testREFF(int& p) {
    int a = 12;
    p = a;

}
void main()
{
    int a = 10;
    int* b = &a;
    testPTR(b);//改变指针指向，但是没改变指针的所指的内容
    cout << a << endl;// 10
    cout << *b << endl;// 10
```

```
41          cout << a << endl;//12
42      }
43
```

在编译器看来, int a = 10; int &b = a; 等价于 int * const b = &a; 而 b = 20; 等价于 *b = 20; 自动转换为指针和自动解引用.

4、在传递函数参数时，什么时候该使用指针，什么时候该使用引用呢?

- 需要返回函数内局部变量的内存的时候用指针。使用指针传参需要开辟内存，用完要记得释放指针，不然会内存泄漏。而返回局部变量的引用是没有意义的
- 对栈空间大小比较敏感（比如递归）的时候使用引用。使用引用传递不需要创建临时变量，开销要更小
- 类对象作为参数传递的时候使用引用，这是C++类对象传递的标准方式

5、堆和栈的区别

- 申请方式不同。
    - 栈由系统自动分配。
- 堆是自己申请和释放的。
- 申请大小限制不同。
    - 栈顶和栈底是之前预设好的，栈是向栈底扩展，大小固定，可以通过ulimit -a查看，由ulimit -s修改。
    - 堆向高地址扩展，是不连续的内存区域，大小可以灵活调整。
- 申请效率不同。

- 堆由程序员分配，速度慢，且会有碎片。

栈空间默认是4M, 堆区一般是 1G - 4G

	堆	栈
管理方式	堆中资源由程序员控制（容易产生memory leak）	栈资源由编译器自动管理，无需手工控制
内存管理机制	系统有一个记录空闲内存地址的链表，当系统收到程序申请时，遍历该链表，寻找第一个空间大于申请空间的堆结点，删除空闲结点链表中的该结点，并将该结点空间分配给程序（大多数系统会在这块内存空间首地址记录本次分配的大小，这样delete才能正确释放本内存空间，另外系统会将多余的部分重新放入空闲链表中）	只要栈的剩余空间大于所申请空间，系统为程序提供内存，否则报异常提示栈溢出。（这一块理解一下链表和队列的区别，不连续空间和连续空间的区别，应该就比较好理解这两种机制的区别了）
空间大小	堆是不连续的内存区域（因为系统是用链表来存储空闲内存地址，自然不是连续的），堆大小受限于计算机系统中有效的虚拟内存（32bit 系统理论上是4G），所以堆的空间比较灵活，比较大	栈是一块连续的内存区域，大小是操作系统预定好的，windows下栈大小是2M（也有是1M，在 编译时确定，VC中可设置）
碎片问题	对于堆，频繁的new/delete会造成大量碎片，使程序效率降低	对于栈，它是有点类似于数据结构上的一个先进后出的栈，进出一一对应，不会产生碎片。（看到这里我突然明白了为什么面试官在问我堆和栈的区别之前先问了我栈和队列的区别）
生长方向	堆向上，向高地址方向增长。	栈向下，向低地址方向增长。

分配方式	堆都是动态分配（没有静态分配的堆）	栈有静态分配和动态分配，静态分配由编译器完成（如局部变量分配），动态分配由alloca函数分配，但栈的动态分配的资源由编译器进行释放，无需程序员实现。
分配效率	堆由C/C++函数库提供，机制很复杂。所以堆的效率比栈低很多。	栈是其系统提供的数据结构，计算机在底层对栈提供支持，分配专门 寄存器存放栈地址，栈操作有专门指令。

形象的比喻

栈就像我们去饭馆里吃饭，只管点菜（发出申请）、付钱、和吃（使用），吃饱了就走，不必理会切菜、洗菜等准备工作和洗碗、刷锅等扫尾工作，他的好处是快捷，但是自由度小。

堆就象是自己动手做喜欢吃的菜肴，比较麻烦，但是比较符合自己的口味，而且自由度大。

6、你觉得堆快一点还是栈快一点?

毫无疑问是栈快一点。

因为操作系统会在底层对栈提供支持，会分配专门的寄存器存放栈的地址，栈的入栈出栈操作也十分简单，并且有专门的指令执行，所以栈的效率比较高也比较快。

而堆的操作是由C/C++函数库提供的，在分配堆内存的时候需要一定的算法寻找合适大小的内存。并且获取堆的内容需要两次访问，第一次访问指针，第二次根据指针保存的地址访问内存，因此堆比较慢。

7、区别以下指针类型?

```
int (*p)[10]
int *p(int)
int (*p)(int)
```

- int *p[10]表示指针数组，强调数组概念，是一个数组变量，数组大小为10，数组内每个元素都是指向int类型的指针变量。
- int (*p)[10]表示数组指针，强调是指针，只有一个变量，是指针类型，不过指向的是一个int类型的数组，这个数组大小是10。
- int *p(int)是函数声明，函数名是p，参数是int类型的，返回值是int *类型的。
- int (*p)(int)是函数指针，强调是指针，该指针指向的函数具有int类型参数，并且返回值是int类型的。

8、new / delete 与 malloc / free的异同

相同点

- 都可用于内存的动态申请和释放

不同点

- 前者是C++运算符，后者是C/C++语言标准库函数
- new自动计算要分配的空间大小，malloc需要手工计算
- new是类型安全的，malloc不是。例如：

```
int *p = new float[2]; //编译错误
int *p = (int*)malloc(2 * sizeof(double));//编译无错误
```

- new调用名为operator new的标准库函数分配足够空间并调用相关对象的构造函数，delete对指针所指对象运行适当的析构函数；然后通过调用名为operator

- new是封装了malloc，直接free不会报错，但是这只是释放内存，而不会析构对象

9、new和delete是如何实现的?

- new的实现过程是：首先调用名为operator new的标准库函数，分配足够大的原始为类型化的内存，以保存指定类型的一个对象；接下来运行该类型的一个构造函数，用指定初始化构造对象；最后返回指向新分配并构造后的的对象的指针
- delete的实现过程：对指针指向的对象运行适当的析构函数；然后通过调用名为operator delete的标准库函数释放该对象所用内存

10、malloc和new的区别?

- malloc和free是标准库函数，支持覆盖；new和delete是运算符，支持重载。
- malloc仅仅分配内存空间，free仅仅回收空间，不具备调用构造函数和析构函数功能，用malloc分配空间存储类的对象存在风险；new和delete除了分配回收功能外，还会调用构造函数和析构函数。
- malloc和free返回的是void类型指针（必须进行类型转换），new和delete返回的是具体类型指针。

> update1:感谢微信好友"猿六学算法"指出错误，已修正！

11、既然有了malloc/free，C++中为什么还需要new/delete呢?直接用malloc/free不好吗?

- malloc/free和new/delete都是用来申请内存和回收内存的。
- 在对非基本数据类型的对象使用的时候，对象创建的时候还需要执行构造函数，销毁的时候要执行析构函数。而malloc/free是库函数，是已经编译的代码，所以不能把构造函数和析构函数的功能强加给malloc/free，所以new/delete是必不可少的。

12、被free回收的内存是立即返还给操作系统吗?

不是的，被free回收的内存会首先被ptmalloc使用双链表保存起来，当用户下一次申请内存的时候，会尝试从这些内存中寻找合适的返回。这样就避免了频繁的系统调用，占用过多的系统资源。同时ptmalloc也会尝试对小块内存进行合并，避免过多的内存碎片。

13、宏定义和函数有何区别?

- 宏在预处理阶段完成替换，之后被替换的文本参与编译，相当于直接插入了代码，运行时不存在函数调用，执行起来更快；函数调用在运行时需要跳转到具体调用函数。
- 宏定义属于在结构中插入代码，没有返回值；函数调用具有返回值。
- 宏定义参数没有类型，不进行类型检查；函数参数具有类型，需要检查类型。
- 宏定义不要在最后加分号。

14、宏定义和typedef区别?

- 宏主要用于定义常量及书写复杂的内容；typedef主要用于定义类型别名。
- 宏替换发生在编译阶段之前，属于文本插入替换；typedef是编译的一部分。
- 宏不检查类型；typedef会检查数据类型。
- 宏不是语句，不在在最后加分号；typedef是语句，要加分号标识结束。
- 注意对指针的操作，typedef char * p_char和#define p_char char *区别巨大。

15、变量声明和定义区别?

- 声明仅仅是把变量的声明的位置及类型提供给编译器，并不分配内存空间；定义要在定义的地方为其分配存储空间。

16、strlen和sizeof区别?

- sizeof是运算符，并不是函数，结果在编译时得到而非运行中获得；strlen是字符处理的库函数。
- sizeof参数可以是任何数据的类型或者数据（sizeof参数不退化）；strlen的参数只能是字符指针且结尾是'\0'的字符串。
- 因为sizeof值在编译时确定，所以不能用来得到动态分配（运行时分配）存储空间的大小。

```
int main(int argc, char const *argv[]){

    const char* str = "name";

    sizeof(str); // 取的是指针str的长度，是8
    strlen(str); // 取的是这个字符串的长度，不包含结尾的 \0。大小是4
    return 0;
}
```

16.2、（补充题）一个指针占多少字节?

在16题中有提到sizeof（str）的值为8，是在64位的编译环境下的，指针的占用大小为8字节；

而在32位环境下，指针占用大小为4字节。

一个指针占内存的大小跟编译环境有关，而与机器的位数无关。

还有疑问的，可以自行打开Visual Studio编译器自己实验一番。

> 感谢微信好友"轻与重"提出补充，已采纳！-2021.07.01

17、常量指针和指针常量区别?

- 指针常量是一个指针，读成常量的指针，指向一个只读变量，也就是后面所指明的int const 和 const int，都是一个常量，可以写作int const *p或const int *p。
- 常量指针是一个不能给改变指向的指针。指针是个常量，必须初始化，一旦初始化完成，它的值（也就是存放在指针中的地址）就不能在改变了，即不能中途改变指向，如int *const p。

> update1:https://www.nowcoder.com/discuss/597948 ，网友“ 牛客191489444号 ”指出笔误，感谢！
> update2:《C++ Primer 5th》 P56页有明确说明常量指针和指针常量，阿秀特意去确认了-20210521。
> 多说一句，网上关于指针常量和常量指针的说法很多跟书本上都不一致，甚至百度百科上跟《C++ Primer 5th》书上在指针常量和常量指针的说法刚好相反，鉴于百度百科是人人都可以去编辑，因此我信书。
> 也希望各位遇到问题时要多去查阅资料，多去确认，不要因为某些博客或者文章说了就确认无疑。

18、a和&a有什么区别?

假设数组int a[10]; int (*p)[10] = &a;其中：

- a是数组名，是数组首元素地址，+1表示地址值加上一个int类型的大小，如果a的值是0x00000001，加1操作后变为0x00000005。*(a + 1) = a[1]。
- &a是数组的指针，其类型为int (*)[10]（就是前面提到的数组指针），其加1时，系统会认为是数组首地址加上整个数组的偏移（10个int型变量），值为数组a尾元素后一个元素的地址。
- 若(int *)p ，此时输出 *p时，其值为a[0]的值，因为被转为int *类型，解引用时按照int类型大小来读取。

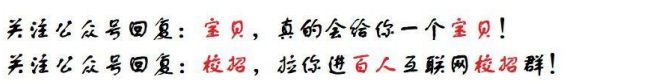
19、C++和Python的区别

- Python是一种脚本语言，是解释执行的，而C++是编译语言，是需要编译后在特定平台运行的。python可以很方便的跨平台，但是效率没有C++高。
- Python使用缩进来区分不同的代码块，C++使用花括号来区分
- C++中需要事先定义变量的类型，而Python不需要，Python的基本数据类型只有数字，布尔值，字符串，列表，元组等等
- Python的库函数比C++的多，调用起来很方便

20、C++和C语言的区别

- C++中new和delete是对内存分配的运算符，取代了C中的malloc和free。
- 标准C++中的字符串类取代了标准C函数库头文件中的字符数组处理函数（C中没有字符串类型）。
- C++中用来做控制态输入输出的iostream类库替代了标准C中的stdio函数库。
- C++中的try/catch/throw异常处理机制取代了标准C中的setjmp()和longjmp()函数。
- 在C++中，允许有相同的函数名，不过它们的参数类型不能完全相同，这样这些函数就可以相互区别开来。而这在C语言中是不允许的。也就是C++可以重载，C语言不允许。
- C++语言中，允许变量定义语句在程序中的任何地方，只要在是使用它之前就可以；而C语言中，必须要在函数开头部分。而且C++不允许重复定义变量，C语言也是做不到这一点的
- 在C++中，除了值和指针之外，新增了引用。引用型变量是其他变量的一个别名，我们可以认为他们只是名字不相同，其他都是相同的。
- C++相对与C增加了一些关键字，如：bool、using、dynamic_cast、namespace等等

> 感谢微信好友“铁锤哥哥”勘误：C++允许重复定义变量->不允许-2021.06.28



在 GitHub 上编辑此页

上次更新: 8/9/2023 贡献者: 阿秀

基础语法-21-40 →