c++编程规范试题

一、bool 与 float 指针变量 与 ”零值“比较的语句

1、bool flag

```
if(flag) // flag is true
if(!flag) // flag is false
```

2、float x

```
const double esp = 1.0E-18; // 精度
if ((x >= -esp) && ( x <= esp))
{ ... }
```

3、char *p

```
if(p == NULL)
if(p != NULL)
```

二、32位 C++程序 sizeof的值

1、

```
char str[] = "Hello";
char *p = str;
char  n = 10;
// 请计算
// sizeof(str) = 4  char类型数组 str指向第一个数据 为指针
// sizeof( p ) = 4  指针类型 4字节 （32位）
// sizeof( n ) = 2
```

2、

```
void Func( char str[100] )
{
  // sizeof(str) = 4
}
```

3、

```
void *p = malloc( 100 );
// sizeof( p ) = 4
```

三、简答题

1、头文件中的 ifndf/define/endif 干什么用

```
一种防止重复定义和重复包含头文件的宏，防止重定义错误和重复引用。
```

2、#include <filename.h> 和 #include "filename.h"

```
#include <filename>
// 尖括号用于引用标准库文件
#include "filename"
// 双引号 引用非标准库头文件
```

3、const 用途

```
const 用于定义常量，编译器可以对其进行数据静态类型安全检查
const 修饰函数的形参，使得传入参数不可被函数内部改变，权限为只读。
const 修饰类的成员函数，任何不会修改数据成员的的成员函数应该用const修饰，该函数 对成员数据权 限为只读，不可写。
const 修饰函数返回值 使得返回值不能为左值
```

4、c++中调用被编译后的函数 为什么要加 extern ”C“ 的声明

```
C++的函数重载机制，使得c++编译器在编译时会产生函数内部标识符，导致编译后的函数名字改变以支持重载和类型安全连接。extern “C” 为C连接交换指定符号，告诉编译器 被修饰的函数为C连接，应该在编译后的库中寻找_<functionName>的 函数。
```

5、请阐述下面两个for循环的优缺点

第一个

```
for (int i = 0; i < N; i++)
{
  if ( condition )
    Do1();
  else
    Do2();
}
```

```
优点：代码简洁
缺点：if语句打断了循环，编译器无法优化循环，当循环次数暴增时，每次循环都要判断，效率较低
```

第二个

```
if (condition)
{
 for(...) { Do1() }
}
else{
 for(...) { Do2() }
}
```

```
优点：先判断再进入循环，循环执行效率高
缺点：程序不够简洁
```

四、关于内存

1、 在函数*void* GetMemory(*char* *p) 中 编译器为p的副本_p分配新的内存， _p指向新的内存地址，但是p没有改变，所以函数GetMemory没有输出，而且分配内存后没有free，会导致内存泄露。

2、 函数 GetMemory return返回了了一个指向栈内存的指针，栈内存在函数结束时消亡了。导致str 的内容是垃圾

3、没有free 导致内存泄露

4、str 分配的内存被free掉了，但是str指针指向的地址没有变，并不是NULL，因此if语句没有起到防御作用。

五、编写strcpy函数

1、不调用string库编写 strcpy

```
#include <iostream>
#include <assert.h>

char* strcpy(char* strDest, const char* strSrc)
{
  // 使用断言
  assert((strDest != NULL) && (strSrc != NULL));
  char * strDestCopy = strDest;
  while( *(strSrc) != '\0'){
    *strDest = *strSrc; // 遍历原字符串赋值
    strSrc++;
    strDest++;
```

```
  }
  *strDest = '\0'; // 补充'\0'

  return strDestCopy;
}

int main()
{
    char str1[6] = "Hello";
    char str2[20];
    strcpy(str2,str1);
    std::cout << "str1 = " << str1 << std::endl;
    std::cout << "str2 = " << str2 << std::endl;

    return 0;
}
```

2、为了增加灵活性，如支持链式表达。

六、编写string 构造函数 析构函数 拷贝赋值

```
// 普通构造函数
String::String(const char* str = NULL)
{
  if(str == NULL){
    m_data = new char[1];
    *m_data = '\0';
  }
  else{
    int length = strlen(str);
    m_data = new char[length + 1];
    strcpy(m_data, str);
  }
}

// 拷贝构造
String::String(const String& other)
{
  int length = strlen(other.m_data);
  m_data = new char[length + 1];
  strcpy(m_data, other.m_data);
}

// 析构函数
String::~String(void)
{
```

```
  delete [] m_data;
}

String& String::operator=(const String& other)
{
  // 1、检查自我赋值
  if(this == &other)
    return *this;
  // 2、释放原来的内存
  delete [] m_data;
  // 3、分配内存，并复制内容
  int length = strlen(other.m_data);
  m_data = new char[length + 1];
  strcpy(m_data, other.m_data);
  // 4、返回
  return *this;
}
```