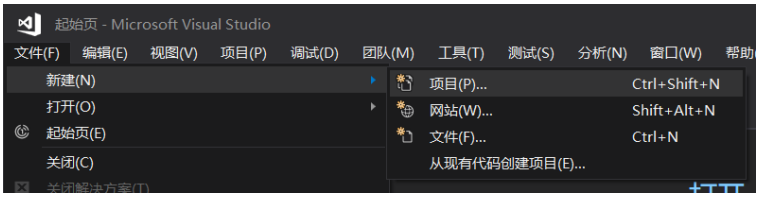
Simple Circuit 项目报告

学院：数据科学与计算机学院　　专业：软件工程
姓名：张伟焜　　学号：17343155

代码运行方法

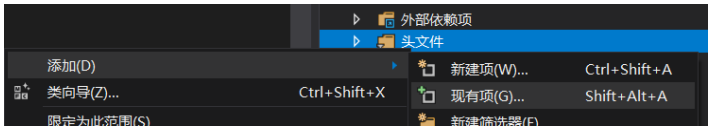
代码编写环境：Win10　Visual Studio2017

打开 visual studio 点击 文件->新建->项目

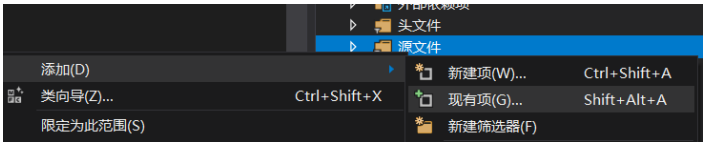


选择 visual C++ 空项目

在右侧资源管理器中 右击头文件 添加现有项， 将 code 两个头文件添加进项目



在右侧资源管理器中 右击源文件 添加现有项， 将 code 中两个源文件添加进项目

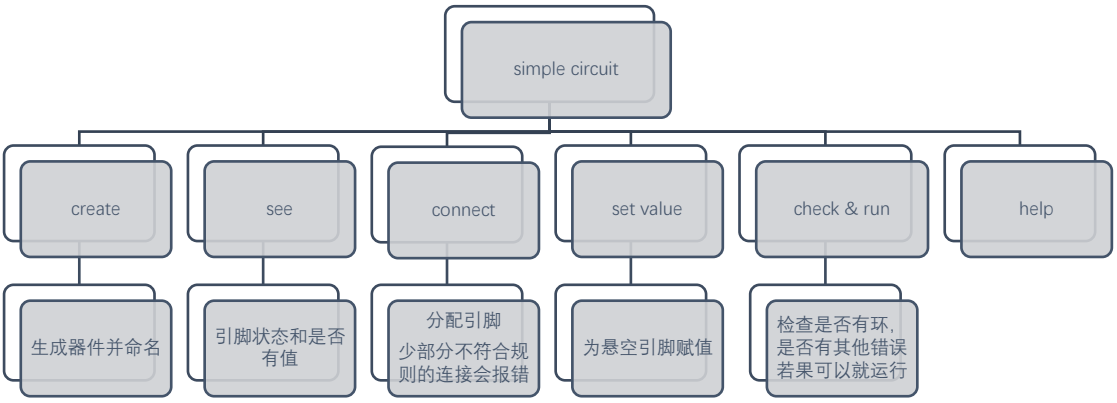


最后点击上方按钮 

运行

刚发现，若头文件和源文件（.h 和.cpp）不在一个文件夹中，则 vs 会显示 #include“gate.h”错误　（上次的 nethake 上交时把.h 和.cpp 分开放在 include 和 src 文件中，所以可能会报错，解决方法:创建项目前,把所有代码放在一个文件中.　为上次期中项目的失误，抱歉）

项目层次



【create】
根据用户输入，产生对应的器件，并将 Node 和 class 匹配，自动生成器件名和引脚名。
【see】
显示引脚状态，显示引脚的当前值。
【connect】
格式————【pin name+pin name】(There is no space around '+'! Press -1 to end)
按指令连接器件，其中一些操作，会抛出异常。如：输出与输出连接，自身输入和自身输出相连；如果指令格式错误或器件名错误或器件不存在也会抛出异常。
【set value】
格式————【pin name=X】(X=0 or 1.There is no space around '='! Press -1 to end)
为当前悬空引脚赋值。如果指令格式错误或器件名错误或器件不存在会抛出异常。
赋值为 0 表示低电平，赋值为 1 或任意非零整数表示高电平。
注意：为实现对一组器件的重复测试，程序设定，在第一个赋值操作开始时，将所有器件的当前值都重置。即，每次赋值操作都会覆盖所有器件之前的值。
【check & run】
对于所有器件，输入引脚不悬空，或悬空但被赋值，即可进行下一步检测。
检查是否有环，并运行。
思路：对 library 中所有门 只有当两个输入引脚都被赋值，该门才能进行运算，运行后将结果赋给下一个门。 以这种方式，在 while 中循环一次，至少有一个门会进行运算并为下个门赋值，while 的最多循环次数与门个数相同。即，while 循环次数大于门的总个数时就出现了环，报错 ! ! !
运行完后，可以按 0 查看每个门的每个引脚的值
注：运行结束后，可返回主菜单，再选择 set value 进行另一组赋值，再运行。

设计思路及项目要求的体现

看到项目要求，首先想到 C++封装的特性，其次想到继承。即，首先声明一个抽象类 Gate 然后各种门都由此派生出来。同时，为了记录并存放（用 vector 存放）每个器件，并且为广度优先的遍历提供可能，再创建一个 Node 结构体，一个节点中放一个类对象，并且有两个 in 指针，一个藕汤指针。(因为 not 门只有一个输入，所以要在其类中单独声明一个构造函数，处理 in2)。

考虑如何让用户对引脚进行操作时，自然想到期间和引脚的统一命名。代码如下：

```
void creat_name(Node* t, string s, int n) {
    t->name = s + to_string(n);
    if (s != "not") {
        t->pin1 = t->name + "_in1";
        t->pin2 = t->name + "_in2";
    }
    else {
        t->pin1 = t->name + "_in";
        t->pin2 = t->name + "_in";
    }
    t->pinout = t->name + "_out";
   }
```

有了统一命名的引脚，用户可以通过输入指令来进行连接和赋值。

在进行全加器的测试时，发现有些输入是并联在一起的，所以创建 cross 类，使用 map，进行匹配。代码如下：

```
class cross {
public:
    map<Node*, vector<Node*>> dot1; //记录与输入引脚1 相并连的输入引脚
    map<Node*, vector<Node*>> dot2; //记录与输入引脚2 相并连的输入引脚
   };
```

同时发现，有些输出接到多个输入，所以将 Node 中的 Node* out 换成 vector<Node*> out

处理异常部分，考虑两大方面：1.指令是否合法，器件是否存在 2.连接与赋值操作是否非法

检测环，原本准备采用广度优先进行环的检测，但是在实际操作过程中发现环的检测可以分为几部分：1.在连线时，将一个器件输出与自身输入相连必定会产生环。2.两个输出相连是非法操作 3.其余产生环的操作，在特定的运行条件下会使程序进入死循环。

其中，1 和 2 在连线时即可检测出环并及时抛出异常；3 在 check & run 时进行检测，在特定的运行条件下（对 library 中所有门，只有当两个输入引脚都被赋值，该门才能进行运算，运行后将结果赋给下一个门。 以这种方式，在 while 中循环一次，至少有一个门会进行运算并为下个门赋值，while 的最多循环次数与门个数相同。即，while 循环次数大于门的总个数时就出现了环，报错！！！），while 的循环次数大于器件总数即可判定程序出现死循环，电路中一定产生了环。

【运算符重载】：对 class Gate 进行<< 的重载，实现输出功能

```
friend ostream& operator<<(ostream &os, const Gate* s) {
    os << s->out;
    return o;
}
```

【继承】： 抽象类 Gate，每种门都公有继承 Gate 类

```
class gate_and :public Gate {
public:
    bool output();
};
class gate_or :public Gate {
public:
    bool output();
};
class gate_not :public Gate {
public:
    gate_not();
    bool output();
};
class gate_exclusive_or :public Gate {
public:
    bool output();
};
class gate_exclusive_nor :public Gate {
public:
    bool output();
};
class gate_nand :public Gate {
public:
    bool output();
};
class gate_nor :public Gate {
public:
    bool output();
};
```

【多态】：运行时多态，虚函数

```
class Gate {
public:
    Gate();
    virtual bool output() = 0;
    virtual bool ifrun();
    virtual void set_node(Node* x);
    virtual Node* get_node();
    virtual bool get_out();
    virtual void set_in1(bool x);
    virtual bool get_in1();
    virtual void set_in2(bool x);
    virtual bool get_in2();
    virtual void set_seted1(bool x);
    virtual bool get_seted1();
    virtual void set_seted2(bool x);
    virtual bool get_seted2();
    virtual void set_seted3(bool x);
    virtual bool get_seted3();
    virtual void set_have_value1(bool x);
    virtual bool get_have_value1();
    virtual void set_have_value2(bool x);
    virtual bool get_have_value2();
    virtual int get_in_port();
    friend ostream& operator<<(ostream &o, const Gate* s) {
        o << s->out;
        return o;
    }
protected:
    bool out;
    bool in_1;
    bool in_2;
    int in_port;
    bool seted1;  //输入1引脚是否悬空
    bool seted2;  //输入2引脚是否悬空
    bool seted3;  //输出引脚是否悬空
    bool have_value1;   //输入1 是否有值
    bool have_value2;   //输入2 是否有值
    Node* device;
};

Gate* cptr = nullptr;
Node* nptr = nullptr;
```

【异常】：throw runtime_error("xxxxxxxxxxx");

输入指令非法时抛出异常：

```
            if (cmd.size()>1)                                                    //
“throw runtime_error” 检测cmd输入和num输入的规范性
                throw runtime_error("Your command is illegal!!!");
            if (cmd[0] - '0'<0 || cmd[0] - '0'>7)
                throw runtime_error("Your command is illegal!!!");
```

输入指令格式错误时抛出异常：

```
            pos = line.find('+');
            if (pos <= 0)
                throw runtime_error("Check your command's format!!!");  //检查指令格式
```

输入的引脚名称不正确或不存在时抛出异常

```
            pos = port1.find('_');    //检查门的名称或格式
            if (pos <= 0)
                throw runtime_error("Check your command's format or pin's name!!!");
            if (n1 == nullptr || n2 == nullptr)       //检查该门是否存在
                throw runtime_error("Check your pin's name!!!");
```

运行条件不满足时抛出异常

```
    if (!library[i].obj->ifrun()) {   //检查是否存在引脚悬空 和 悬空引脚是否被赋值
                cout << "There are some pins with no input(s). Please connect them or
set inputs." << endl << endl;
                throw 1;
            }
```

连接错误或出现环时抛出异常

```
                    if (n1->obj->get_seted1() == false) {
                        cdot.dot1[n1].push_back(n2);
                        n1->in1_cross = true;
                    }
                    else {
                        throw runtime_error("Something wrong!!!");    //不允许a+b
b+c c+d ,只能a+b a+c a+d （其中 a,b,c,d均为输入引脚）
                    }

                if (port1.back() == 't'&& port2.back() == 't')
                    throw  runtime_error("WARNING:you  can't  put  two  \"out\"  pins
together!!!");   //两个输出不能相连

                if (n1 == n2)
                    throw runtime_error("WARNING:there might be an Infinite loop!!!");
//自身输入和输出不能相连

            if (ctime>library.size() + 1)
                throw runtime_error("The design is wrong. Infinite loop!!!!");
```

【封装】：Gate 抽象类作为基，派生各种类型的门，每个 class 放在一个 Node 里，Node 存在 vector 中

拓展

1.门的种类扩展：异或、同或、与非、或非门
2.检测输入的指令是否正确，如有错误及抛出异常。如，在主界面输入字母，或 7、8、9 等非法指令
3.连接或赋值时，检测引脚名称是否正确，检测该引脚（器件）是否存在
4.用 class cross 实现了输入与输入的连接，如，全加器中“exclusiveOR1_in1+and2_in1”，可以实现赋值时 exclusiveOR1_in1 和 and2_in1 同步赋值。
5.可以在运行后再次赋值，提高程序可重复性。

测试样例

【测试所用的指令已放在“command”中】

1.全加器（可重复赋值，重复测试，共八组数据）
2.环的检测（三组）

不足及改进空间

1.因时间关系，没能实现器件的删除，重连
2.没有实现图形界面

【写在最后】
由于临近期末，报告可能写的不太详细，很抱歉给您造成不便。
有任何操作上的问题可以邮件联系我：zhangwk8@mail2.sysu.edu.cn
QQ：501352226