

# C++程序设计(拾伍)

---

徐东/计算数学

# 内容

- 创建自定义的新数据类型 (II)
  - 类 `class`

# 任务肆拾柒

- 处理 89 位学生的个人信息
  - 姓名
  - 性别
  - 年龄
  - 身高
  - 体重

# 任务肆拾柒

- 学生的个人信息

- 姓名
- 性别
- 年龄
- 身高
- 体重

- 一切皆是数据（类型）

```
struct Student{  
    string name;  
    char   gender;  
    int    age;  
    double weight;  
    double height;  
};
```

# 任务肆拾柒：输入数据

```
struct Student{
    string name;
    char   gender;
    int    age;
    double weight;
    double height;
};

int main(){
    const int NUMBER = 89;
    Student students[NUMBER];

    //依次输入每位学生的个人信息
    for(int i=0;i<NUMBER;++i){
        cin >> students[i].name >> students[i].gender
            >> students[i].age  >> students[i].height
            >> students[i].weight;
    }

    return 0;
}
```

# 任务肆拾柒：输入数据

```
int main(){  
    const int NUMBER = 89;  
    Student students[NUMBER];  
  
    //依次输入每位学生的个人信息  
    for(int i = 0; i < NUMBER; ++i){  
        cin >> students[i].name >> students[i].gender  
            >> students[i].age >> students[i].height  
            >> students[i].weight;  
    }  
    return 0;  
}
```

- 声明整型常量（数组大小）

# 任务肆拾柒：输入数据

```
int main(){
    const int NUMBER = 89;
    Student students[NUMBER];

    //依次输入每位学生的个人信息
    for(int i = 0; i < NUMBER; ++i){
        cin >> students[i].name >> students[i].gender
            >> students[i].age >> students[i].height
            >> students[i].weight;
    }
    return 0;
}
```

- 声明 **Student**数组 students

# 任务肆拾柒：输入数据

```
int main(){
    const int NUMBER = 89;
    Student students[NUMBER];

    //依次输入每位学生的个人信息
    for(int i = 0; i < NUMBER; ++i){
        cin >> students[i].name >> students[i].gender
            >> students[i].age >> students[i].height
            >> students[i].weight;
    }

    return 0;
}
```

- 通过 for 循环遍历整个数组 students



# 任务肆拾柒：输入数据

```
int main(){  
    const int NUMBER = 89;  
    Student students[NUMBER];  
  
    //依次输入每位学生的个人信息  
    for(int i = 0; i < NUMBER; ++i){  
        cin >> students[i].name >> students[i].gender  
            >> students[i].age >> students[i].height  
            >> students[i].weight;  
    }  
    return 0;  
}
```

- **数组元素** `students[i]` 是 **Student** 类型

# 任务肆拾柒：输入数据

```
int main(){  
    const int NUMBER = 89;  
    Student students[NUMBER];  
  
    //依次输入每位学生的个人信息  
    for(int i = 0; i < NUMBER; ++i){  
        cin >> students[i].name >> students[i].gender  
            >> students[i].age >> students[i].height  
            >> students[i].weight;  
    }  
    return 0;  
}
```

- 通过点运算符 (.) 对数组元素的每个成员分别进行处理

# 任务肆拾柒：输入数据

```
int main(){  
    const int NUMBER = 89;  
    Student students[NUMBER];  
  
    //依次输入每位学生的个人信息  
    for(int i = 0; i < NUMBER; ++i){  
        cin >> students[i].name >> students[i].gender  
            >> students[i].age >> students[i].height  
            >> students[i].weight;  
    }  
    return 0;  
}
```

- **cin 语句太复杂**

# 任务肆拾柒：输入数据

```
int main(){  
    const int NUMBER = 89;  
    Student students[NUMBER];  
  
    //依次输入每位学生的个人信息  
    for(int i = 0; i < NUMBER; ++i){  
        cin >> students[i].name >> students[i].gender  
            >> students[i].age >> students[i].height  
            >> students[i].weight;  
    }  
    return 0;  
}
```

- **cin 语句太复杂 (>> 只能处理基本数据类型的输入)**

# 任务肆拾柒：输入数据

```
int main(){  
    const int NUMBER = 89;  
    Student students[NUMBER];  
  
    //依次输入每位学生的个人信息  
    for(int i = 0; i < NUMBER; ++i){  
        cin >> students[i].name >> students[i].gender  
            >> students[i].age >> students[i].height  
            >> students[i].weight;  
    }  
    return 0;  
}
```

- **cin 语句太复杂（运算符重载 >>）**

# 运算符重载

```
istream &operator >>(istream &input, Student &one){  
    input >> one.name >> one.gender >> one.age  
        >> one.height >> one.weight;  
    return input;  
}
```

```
int main(){  
    const int NUMBER = 3;  
    Student students[NUMBER];  
  
    //依次输入每位学生的个人信息  
    for(int i=0;i<NUMBER;++i){  
        cin >> students[i];  
    }  
    return 0;  
}
```

```
for(int i=0;i<NUMBER;++i){  
    cin >> students[i].name  
        >> students[i].gender  
        >> students[i].age  
        >> students[i].height  
        >> students[i].weight;  
}
```

# 运算符重载：输入运算符 (>>)

```
istream &operator >>(istream &input, Student &one){  
    input >> one.name >> one.gender >> one.age  
        >> one.height >> one.weight;  
    return input;  
}
```

- **重载输入运算符 >>**

# 运算符重载：输入运算符 (>>)

```
istream &operator >>(istream &input, Student &one){  
    input >> one.name >> one.gender >> one.age  
        >> one.height >> one.weight;  
    return input;  
}
```

- **istream**
  - **返回类型**



# 运算符重载：输入运算符 (>>)

```
istream &operator >>(istream &input, Student &one){  
    input >> one.name >> one.gender >> one.age  
        >> one.height >> one.weight;  
    return input;  
}
```

- &

- 引用

# 运算符重载：输入运算符 (>>)

```
istream &operator >>(istream &input, Student &one){  
    input >> one.name >> one.gender >> one.age  
        >> one.height >> one.weight;  
    return input;  
}
```

- `istream &`
  - 返回引用

# 运算符重载：输入运算符 (>>)

```
istream &operator >>(istream &input, Student &one){  
    input >> one.name >> one.gender >> one.age  
        >> one.height >> one.weight;  
    return input;  
}
```

- operator
  - 关键字
  - 运算符重载的标志

# 运算符重载：输入运算符 (>>)

```
istream &operator >>(istream &input, Student &one){  
    input >> one.name >> one.gender >> one.age  
        >> one.height >> one.weight;  
    return input;  
}
```

- >>

- 被重载的运算符

# 运算符重载：输入运算符 (>>)

```
istream &operator >>(istream &input, Student &one){  
    input >> one.name >> one.gender >> one.age  
        >> one.height >> one.weight;  
    return input;  
}
```

- `istream &input, Student &one`
  - 参数列表
  - 参数均为引用

# 运算符重载：输入运算符 (>>)

```
istream &operator >>(istream &input, Student &one){  
    input >> one.name >> one.gender >> one.age  
        >> one.height >> one.weight;  
    return input;  
}
```

- **函数体**

- 定义如何读取 Student 类型数据（变量）的行为
- 实现运算符 (>>) 新功能的具体代码

# 运算符重载：输入运算符 (>>)

```
istream &operator >>(istream &input, Student &one){  
    input >> one.name >> one.gender >> one.age  
    >> one.height >> one.weight;  
    return input;  
}
```

- **注意**

- 必须将返回类型和形式参数设置为“引用 (&)”
- 形式参数出现的顺序

# 任务肆拾柒：输入数据

```
int main(){
    const int NUMBER = 89;
    Student students[NUMBER];

    .....

    //依次输出每位学生的个人信息
    for(int i = 0; i < NUMBER; ++i){
        cout << students[i].name
              << students[i].gender
              << students[i].age
              << students[i].height
              << students[i].weight << endl;
    }

    return 0;
}
```



# 任务肆拾柒：输入数据

```
int main(){
    const int NUMBER = 89;
    Student students[NUMBER];
    .....

    //依次输出每位学生的个人信息
    for(int i = 0; i < NUMBER; ++i){
        cout << students[i].name
              << students[i].gender
              << students[i].age
              << students[i].height
              << students[i].weight << endl;
    }

    return 0;
}
```



重载输出运算符 >>

# 运算符重载：输出运算符 (<<)

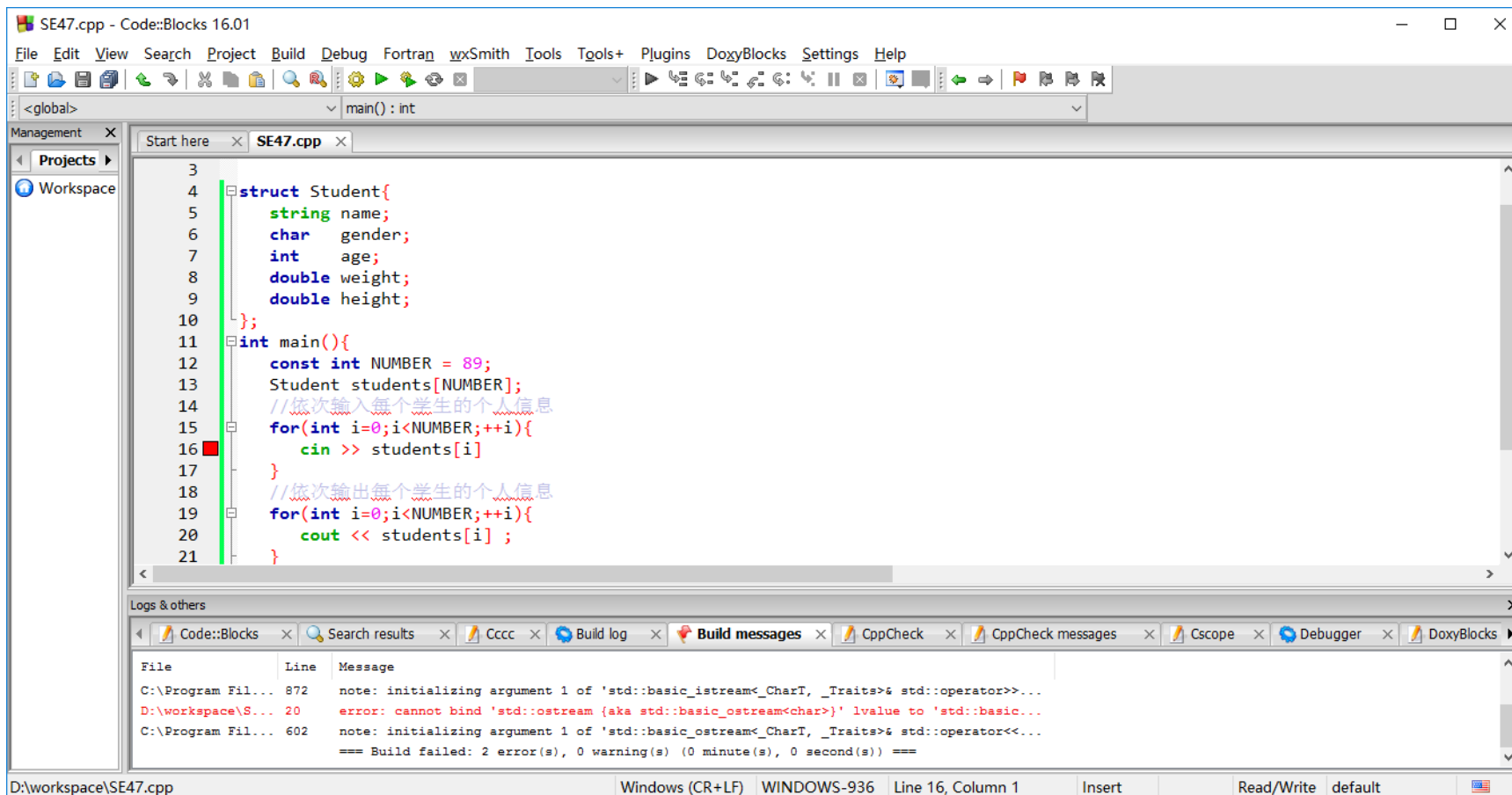
```
ostream &operator <<(ostream &output, const Student &one){
    output << one.name    << "\t" << one.gender << "\t"
           << one.age      << "\t" << one.height << "\t"
           << one.weight << endl;
    return output;
}

int main(){
    const int NUMBER = 89;
    Student students[NUMBER];

    .....

    //依次输出每位学生的个人信息
    for(int i=0;i<NUMBER;++i){
        cout << students[i] ;
    }
    return 0;
}
```

# 运算符重载



- 错误原因：未提供重载的输入运算符和输出运算符

# 任务肆拾捌

- 处理 89 位学生的个人信息
  - 姓名
  - 性别
  - 年龄
  - 身高
  - 体重
  - 身体质量指数 (BMI)

# 任务肆拾捌

```
struct Student{  
    string name;  
    char    gender;  
    int     age;  
    double  weight;  
    double  height;  
    double  bmi;  
};
```

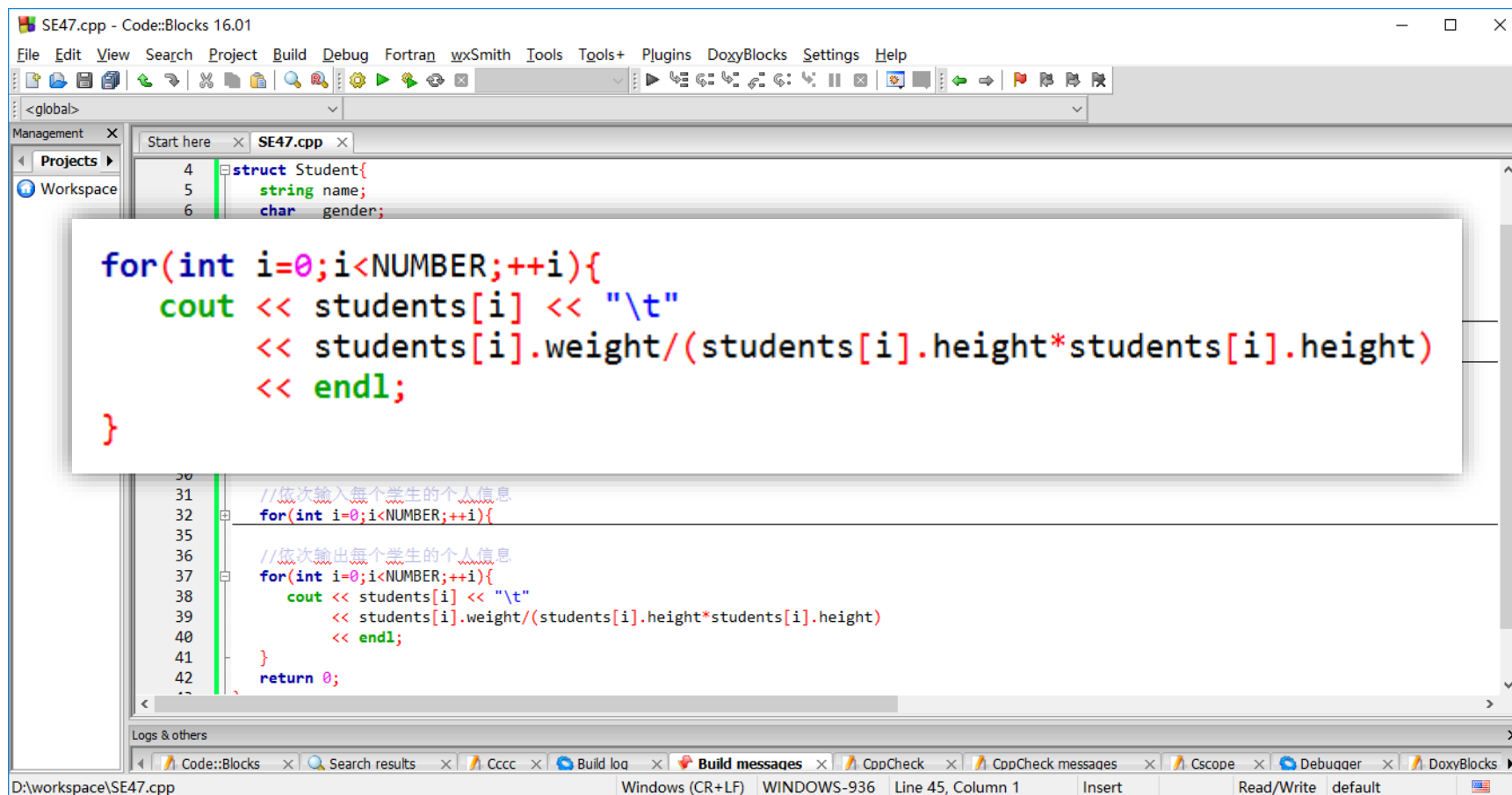
- 是否需要新增成员变量 bmi ?
  - 不需要

# 任务肆拾捌

```
struct Student{  
    string name;  
    char    gender;  
    int     age;  
    double  weight;  
    double  height;  
    double  bmi;  
};
```

- 是否需要新增成员变量 bmi ?
  - 不需要 (通过成员变量 weight 和 height 计算获得)

# 任务肆拾捌



```
SE47.cpp - Code::Blocks 16.01
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help

<global>
Management
  Start here x SE47.cpp x
  Projects
  Workspace
    4 struct Student{
    5     string name;
    6     char gender;

for(int i=0;i<NUMBER;++i){
    cout << students[i] << "\t"
    << students[i].weight/(students[i].height*students[i].height)
    << endl;
}

30
31 //依次输入每个学生的个人信息
32 for(int i=0;i<NUMBER;++i){
35
36 //依次输出每个学生的个人信息
37 for(int i=0;i<NUMBER;++i){
38     cout << students[i] << "\t"
39     << students[i].weight/(students[i].height*students[i].height)
40     << endl;
41 }
42 return 0;

Logs & others
Code::Blocks x Search results x Cccc x Build log x Build messages x CppCheck x CppCheck messages x Cscope x Debugger x DoxyBlocks
D:\workspace\SE47.cpp Windows (CR+LF) WINDOWS-936 Line 45, Column 1 Insert Read/Write default
```

# 任务肆拾捌

- 学生的个人信息
  - 姓名
  - 性别
  - 年龄
  - 身高
  - 体重
  - 身体质量指数 (BMI)
- 更好的解决方案
  - 把计算BMI的函数放入新类型 Student 中



# 类型 Student: 添加计算BMI的函数

```
struct Student{  
    string name;  
    char    gender;  
    int     age;  
    double weight;  
    double height;
```

```
    double calBmi(){  
        return weight / ( height * height );  
    }
```

```
};
```

- **成员函数**

- **专属于数据类型 Student 的函数**

# 类型 Student: 添加计算BMI的函数

```
struct Student{  
    string name;  
    char    gender;  
    int     age;  
    double weight;  
    double height;
```

```
    double calBmi(){  
        return weight / ( height * height );  
    }
```

```
};
```

- **成员函数**
  - **没有参数列表**

# 类型 Student: 添加计算BMI的函数

```
struct Student{  
    string name;  
    char    gender;  
    int     age;  
    double  weight;  
    double  height;
```

```
    double calBmi(){  
        return weight / ( height * height );  
    }
```

```
};
```

- **成员函数**

- **没有参数列表**（调用该函数时，无需提供其他信息。）

# 类型 Student: 添加计算BMI的函数

```
struct Student{  
    string name;  
    char    gender;  
    int     age;  
    double  weight;  
    double  height;
```

```
    double calBmi(){  
        return weight / ( height * height );  
    }
```

```
};
```

- **成员函数**

- 可以直接访问该类型的数据成员和成员函数

# 类型 Student: 添加计算BMI的函数

```
struct Student{  
    string name;  
    char    gender;  
    int     age;  
    double weight;  
    double height;
```

```
    double calBmi(){  
        return weight / ( height * height );  
    }
```

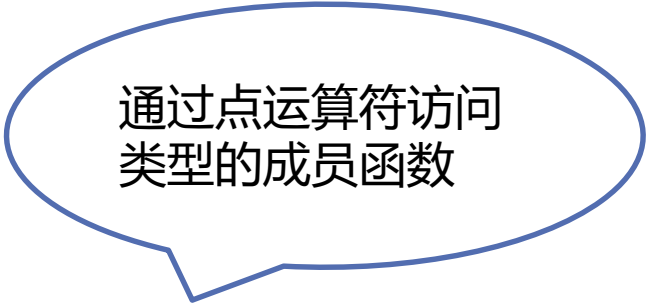
```
};
```

- **成员函数**

- 可以直接访问该类型的数据成员和成员函数

# 类型 Student: 访问成员函数

```
struct Student{  
    string name;  
    char   gender;  
    int    age;  
    double weight;  
    double height;  
    double calBmi(){  
        return weight/(height*height);  
    }  
};
```



通过点运算符访问  
类型的成员函数

```
int main(){  
    Student one ;  
    cin >> one.name >> one.height >> one.weight  
    cout << one.name << "\t" ;  
    cout << one.calBmi() << endl;  
  
    return 0;  
}
```

# 类型 Student

```
int main(){  
    Student one ;  
    cin >> one.name >> one.height >> one.weight;  
    cout << one.name << "\\t" ;  
    cout << one.calBmi() << endl;  
  
    return 0;  
}
```

- 新类型的使用

# 类型 Student

```
int main(){  
    Student one ;  
    cin >> one.name >> one.height >> one.weight;  
    cout << one.name << "\t" ;  
    cout << one.calBmi() << endl;  
  
    return 0;  
}
```

- 新类型的使用
  - 声明变量



# 类型 Student

```
int main(){
    Student one ;
    cin >> one.name >> one.height >> one.weight;
    cout << one.name << "\t" ;
    cout << one.calBmi() << endl;

    return 0;
}
```

- **新类型的使用**
  - **声明变量**
  - **使用变量**

# 类型 Student

```
int main(){
    Student one ;
    cin >> one.name >> one.height >> one.weight;
    cout << one.name << "\t" ;
    cout << one.calBmi() << endl;

    return 0;
}
```

- 新类型的使用
  - 声明变量
  - 使用变量

# 类型 Student

```
int main(){  
    Student one ;  
    cin >> one.name >> one.height >> one.weight;  
    cout << one.name << "\\t" ;  
    cout << one.calBmi() << endl;  
  
    return 0;  
}
```

- 新类型的使用
  - 声明变量
  - 使用变量（通过点运算符访问成员）

# 创建新的数据类型

```
struct Student{  
    string name;  
    char    gender;  
    int     age;  
    double  weight;  
    double  height;  
  
    double  calBmi(){  
        return weight/(height*height);  
    }  
};
```

# 创建新的数据类型

```
struct Student{  
    string name;  
    char    gender;  
    int     age;  
    double  weight;  
    double  height;  
  
    double  calBmi(){  
        return weight/(height*height);  
    }  
};
```

- 自定义数据类型

# 创建新的数据类型

```
struct Student{  
    string name;  
    char    gender;  
    int     age;  
    double  weight;  
    double  height;  
  
    double  calBmi(){  
        return weight/(height*height);  
    }  
};
```

- 自定义数据类型
  - 成员变量

# 创建新的数据类型

```
struct Student{  
    string name;  
    char    gender;  
    int     age;  
    double  weight;  
    double  height;
```

```
    double calBmi(){  
        return weight/(height*height);  
    }
```

```
};
```

- 自定义数据类型

- 成员变量

- 成员函数

- 既包含成员变量又包含成员函数的数据类型统称为 “**类**”

# 任务肆拾捌

- 根据BMI的大小（升序），输出 89 位学生的个人信息。
- 关键
  1. 根据BMI，对学生排序。
  2. sort 函数
    - 不支持 Student 数组的排序
    - 解决



# 任务肆拾捌

- 根据BMI的大小（升序），输出 89 位学生的个人信息。
- 关键
  1. 根据BMI，对学生进行排序。
  2. sort 函数
    - 不支持 Student 数组的排序
    - 解决：提供 sort 函数的第三个参数（排序标准）

# 任务肆拾捌

```
bool cp(Student a, Student b){  
    return a.calBmi() < b.calBmi();  
}
```

```
int main(){  
    const int NUMBER = 89;  
    Student students[NUMBER];  
    .....//依次输入每位学生的个人信息  
  
    sort(students, students + NUMBER, cp);  
  
    .....//依次输出每位学生的个人信息  
  
    return 0;  
}
```

```
struct Student{  
    string name;  
    char   gender;  
    int    age;  
    double weight;  
    double height;  
  
    double calBmi(){  
        return weight/(height*height);  
    }  
};
```

# 任务肆拾捌

```
bool cp(Student a, Student b){  
    return a.calBmi() < b.calBmi();  
}
```

```
int main(){  
    const int NUMBER = 89;  
    Student students[NUMBER];  
    .....//依次输入每位学生的个人信息
```

```
    sort(students, students + NUMBER, cp); //根据BMI升序排序
```

```
    .....//依次输出每位学生的个人信息
```

```
    return 0;
```

```
}
```

```
struct Student{  
    string name;  
    char   gender;  
    int    age;  
    double weight;  
    double height;  
  
    double calBmi(){  
        return weight/(height*height);  
    }  
};
```

**待续.....**