```
/*
W01 CALCULATOR Part A
    Create a program that reads two numbers and a operator and prints the result.

    Example (The text at the back> in each line is read during execution):
    Numbers 1> 20
    Operator> *
    Numbers 2> 5
    The answer of 20 * 5 is 100.
 */

#include <iostream>

using namespace std;

int main() {
   int num1, num2;
   char op;

    cout << "Enter Numbers" << '\n';
    cout << "Number 1: " << '\n';
    cin >> num1;
    cout << "Operator: " << '\n';
    cin >> op;
    cout << "Number 2: " << '\n';
    cin >> num2;

    if (!cin){
        cout << "Enter Only Numbers" << '\n';
    }

    switch (op) {
        case '*':
            cout << "The answer of " << num1 << op << num2 << " is " << num1*num2 << '\n';
            break;

        case '+':
            cout << "The answer of " << num1 << op << num2 << " is " << num1+num2 << '\n';
            break;

        case '-':
            cout << "The answer of " << num1 << op << num2 << " is " << num1-num2 << '\n';
            break;

        case '/':
            if (num2==0) {
                cout << "Cannot divide by 0" << '\n';
            }
            else
            cout << "The answer of " << num1 << op << num2 << " is " << num1/num2 << '\n';
            break;
        default:
            break;
    }
}

/* OUTPUT:
    Enter Numbers
    Number 1: 2
    Operator: *
    Number 2: 4
    The answer of 2*4 is 8
*/
```

```
/*
W01 CALCULATOR Part B
      Create a program that reads a text string with two numbers and a operator and prints the
      result. This text should be interpreted by the program.
      Example (The text behind the > is read during execution) Both versions below should work
      : (not no space in second)

      Enter two numbers with an operator between> 10 + 15
      The answer to 10 + 15 is 25.

      Enter two numbers with an operator between> 10+15
      The answer to 10 + 15 is 25.
 */

#include <iostream>
using namespace std;

int main() {
    int num1, num2 = 0;
    char op;
    cout<< "Enter two numbers with an operator between: "<< flush;
    cin>> num1 >> op >> num2;

    if (!cin) { // check for invalid input.
        cout<<"Only numbers please"<<endl;
    }

    switch (op) {
        case '+':
            cout<<endl<< "The answer to " << num1 << op << num2 << " is " << num1+num2 << endl;
            break;
        case '-':
            cout<<endl<< "The answer to " << num1 << op << num2 << " is " << num1-num2 << endl;
            break;
        case '*':
            cout<<endl<< "The answer to " << num1 << op << num2 << " is " << num1*num2 << endl;
            break;
        case '/':
            if (num2 == 0) { // division by zero
                cout << "Don't divide by zero!";
            }
            else
                cout<<endl<< "The answer to " << num1 << op << num2 << " is " << (double) num1/
num2 << endl;
            break;
        default:
            break;
    }
    return 0;
}

/* OUTPUT:
    Enter two numbers with an operator between: 2+4
    The answer to 2+4 is 6
*/
```

```
/*
W01 CALCULATOR Part C
    Create a program that reads a text string with multiple numbers and operators and prints
    the result. This text should be interpreted by the program.

    Example (The text at the back of the > is read during execution):
    Enter numbers with an operator between> 2 + 3 * 5
    The answer to 2 + (3 * 5) is 17.
*/

#include <iostream>

class calculator {
public:
    double num1, num2, num3, res = 0.0;
    char op1, op2;

    void input() {
        std::cout << "Enter numbers with operators between: " << std::flush;
        std::cin >> num1 >> op1 >> num2 >> op2 >> num3;

        if ((op1 == '/' && num2 == 0) || (op2 == '/' && num3 == 0))
            std::cout << "You can't divide by zero" << std::endl;
        else if ((op2 == '*' || op2 == '/') && (op1 == '-' || op1 == '+')) {
            switch (op2) {
                case '*':
                    res = num2*num3;
                    break;
                case '/':
                    res = num2/num3;
                    break;
                case '%':
                    res = (int)num2 % (int)num3;
                    break;
                default:
                    break;
            }
            switch (op1) {
                case '+':
                    res = num1+res;
                    break;
                case '-':
                    res = num1-res;
                    break;
                default:
                    break;
            }
        }
        else {
            switch (op1) {
                case '+':
                    res = num1+num2;
                    break;
                case '-':
                    res = num1-num2;
                    break;
                case '*':
                    res = num1*num2;
                    break;
                case '/':
                    res = num1/num2;
                    break;
                default:
                    break;
            }
```

```
            switch (op2) {
                case '+':
                    res = res+num3;
                    break;
                case '-':
                    res = res-num3;
                    break;
                case '*':
                    res = res*num3;
                    break;
                case '/':
                    res = res/num3;
                    break;
                default:
                    break;
            }
        }
    }

    void display() {
        if (num3 == '\0') {
            std::cout << "The answer to " << num1 << op1 << num2 << " is " << res << std::endl;
        }
        else std::cout << "The answer to " << num1 << op1 << num2 << op2 << num3 << " is " << r
es << std::endl;
    }
};

int main() {
    calculator Calc;
    Calc.input();
    Calc.display();
    return 0;
}

/* OUTPUT:
    Enter numbers with operators between: 2 + 3 * 5
    The answer to 2+3*5 is 17
*/
```

```
/*
W01 CALCULATOR Part D is an extension of Part C.
    Create a program that reads a text string with multiple numbers and operators and prints
    the result.
    But now it should be possible, among other things, to use () around partial expressions and
    the program must deal with arbitrary number of parameters. It is also advantageous to add
    some other operators %, ^ (power) etc.
    A natural extension is also to create the program so that the text string can be entered at
    the start of the program from the command line.

    Example (The text at the back of the > is read during execution):
    Enter two numbers with an operator between>4 + 5 + (7 + 4) * 3 + 4 * 4 + 9/3 + 4 * 12 - 3/4
    The answer to 4 + 5 + (7 + 4) * 3 + 4 * 4 + 9/3 + 4 * 12 - 3/4 is 108.25.
*/

#include <iostream>
#include <string>
#include <cmath>

#define pi 3.1415

int no_errors = 0;
double error(const char* s) {
    no_errors++;
    std::cerr << "error: " << s << '\n';
    return 1;
}

enum Token_val {
    name, number, end, PLUS='+', MINUS='-', mul='*',
    DIV='/', print=';', assign='=', LP='(' ,RP=')',
    MOD='%', power = '^'
};


// Token: sequence of chars, (kind, value) pairs: i.e. (number, 3)
class Token {
public:
    char kind;
    double value; // for numbers
    Token(char ch): kind(ch),value(0) {}
    Token(char ch, double val): kind(ch), value(val) {}
};

class Token_stream {
public:
    Token_stream();
    Token get();
    void putback(Token curr_token);

private:
    bool full; // token in buffer?
    Token buffer;
};

Token_stream::Token_stream():full(false),buffer(0){}; // constructor

void Token_stream::putback(Token curr_token) {
    if (full) error("putback() into full buffer");
    buffer = curr_token;
    full = true;
}

Token Token_stream::get() { // read chars from cin and compose tokens
    if (full) { // token in buffer
```

```
        full = false;   // read from cin
        return buffer; // return token from buffer
    }
    char ch;
    std::cin >> ch;
    switch (ch) {
        case ';':   // print
        case 'q':   // quit
        case '*': case '/': case '+': case '-':
        case '(': case ')': case '=': case '%':
        case '^':             return Token(ch);
        case '0': case '1': case '2': case '3':
        case '4': case '5': case '6': case '7':
        case '8': case '9': case '.':
        {
            std::cin.putback(ch); // put digit back into input stream
            double val;
            std::cin >> val;
            return Token(number,val);
        }
        default:
            error("Bad token\n");
    }
    return false;
}

//double num_val;
Token_stream ts;
double expression();

/* Parsing (bottom-up)
 4) Expression: must be a term or end with a term
        Term
        Expression '+' term
        Expression '-' term
 3) Term: must be a primary or end with a primary
        Primary
        Term '*' primary
        Term '/' primary
        Term '%' primary
 2) Primary: a number or '(' followed by an expression followed by ')'
        Number
        '(' expression')'
 1) Number:
        floating-point-literal
 */

/* e.g. 45 + 11 * 7
// 45: number -> primary -> term -
> expression followed by '+', 11=term: (expression+term=45+11)
// 11: number -> primary -> term followed by '*', 7=primary: (term * primary = 11 * 7)
// 45+11*7 is an expression where 3)(11*7) will be evaluated before the addition 4)45+(11*7)
*/

double primary() { // handles numbers and parentheses, calls expression().
    Token curr_token = ts.get();
    switch (curr_token.kind) {
        case number: {
            return curr_token.value;
        }
        case MINUS:
            return -primary();
        case LP: {  // handle << '(' expression ')' >>
            auto e = expression();
            curr_token = ts.get();
```

```
            if (curr_token.kind != RP) error("')' missing\n");
            return e;
        }
        default:
            error("Primary expected!\n");
    }
    return false;
}

// multiplication and division has higher precedence than addition and subtraction.
double term() { // handles '*', '/' and '%'. calls primary
    double left = primary();
    Token curr_token = ts.get(); // get next token
    while(true) {
        switch (curr_token.kind) {
                //  Term '*' primary
            case mul:
                left *= term();
                curr_token = ts.get();
                break;


            case MOD: {
                auto d = term();
                int a=left;
                int b=d;
                left = a%b;
                curr_token = ts.get();
                break;
            }
            case power: // use pow not xor
                left = pow(left,term());
                curr_token = ts.get();
                break;

            case DIV:
                if (auto d = term()) {
                    left /= d;
                    curr_token = ts.get();
                    break;
                }
                error("do not divide by 0...\n");
            default:
                ts.putback(curr_token);
                return left;
        }
    }
}

// add and subtract
double expression() {
    double left = term(); // evaluate a Term
    Token curr_token = ts.get(); // get next token
    while (true) {
        switch (curr_token.kind) { // check which kind of token it is.
            case PLUS:
                left += term();
                curr_token = ts.get();
                break;
            case MINUS:
                left -= term();
                curr_token = ts.get();
                break;
            default:
                ts.putback(curr_token); // put token back into token stream
```

```
                return left; // return final answer.
        }
    }
}


int main() {
    double val = 0.0;
    try {
        while (std::cin) {
            std::cout << "Enter expression: \n";
            Token curr_token = ts.get();
            if(curr_token.kind == 'q') break;
            if(curr_token.kind == ';') {
                std::cout<<"=" << val << std::endl;
            }
            else {
                ts.putback(curr_token);
                }
            val = expression();
            }
        }
    catch (std::exception &e) {
        std::cerr<<"E: "<<e.what()<<'\n';
        return 1;
    }
}

/* OUTPUT:
    Enter expression:
    4 + 5 + (7 + 4) * 3 + 4 * 4 + 9/3 + 4 * 12 - 3/4 ;
    Enter expression:
    =108.25
*/
```

```
/*
W02 CLASSES Part A
    Create class Student that stores name, age and what degree they are on.
    In the constructor the must be able to accept information to initialise   the object,and
the constructor must print this to the screen: "Create a Student: <name>, <age>, <Degree>".
    The destructior must print the following message to the screen: "Deleted: : <name>, <age>,
<Degree>".
    Also create a method Print() and set and get method for all data.
 */

#include <iostream>
#include <vector>

//-------------------------------------------------------
class Student {
public:
    // Constructors
    Student();  //-Default constructor
    Student(std::string, int, std::string); //-Overloaded constructor
    ~Student(); //-Destructor
    //-Accessors (const = not modifiable)
    std::string getName() const;    //-return name
    int getAge() const;             //-return age
    std::string getDegree() const;  //-return degree
    //-Mutators (modifiable)
    void setName();     //-modify name
    void setAge();      //-modify age
    void setDegree();   //-modify degree
    // Print method
    void printData();

private:
    std::string Name;
    int Age;
    std::string Degree;
};
//-------------------------------------------------------

//-Default constructor (:: = 'scope resolution')
Student::Student() {
    std::cout << "Create a student: <name>, <age>, <degree> \n";
    Name = "John Doe";
    Age = -1;
    Degree = "Other degree";
}
//-Overloaded constructor
Student::Student(std::string n, int a, std::string d) {
    std::cout << "Create a student: <name>, <age>, <degree> \n";
    Name = n;
    Age = a;
    Degree = d;
}
//-Destructor
Student::~Student() {
    std::cout << "Deleted student: "<< Name <<", " << Age <<", "<< Degree << std::endl;
}
// Print method
void Student::printData() {
    std::cout <<"Created a student: "<< Name <<", " << Age <<", "<< Degree << std::endl;
}
```

```cpp
//-----------Get----------------------
std::string Student::getName() const {
    return Name;
}
int Student::getAge() const {
    return Age;
}
std::string Student::getDegree() const {
    return Degree;
}

//-----------Set----------------------
void Student::setName() {
    std::string n;
    std::cout<<"Enter name: ";
    std::cin>>n;
    Name = n;
}
void Student::setAge() {
    int a;
    std::cout<<"Enter age: ";
    std::cin>>a;
    Age = a;
}
void Student::setDegree() {
    std::string d;
    std::cout<<"Enter degree: ";
    std::cin>>d;
    Degree = d;
}

int main() {
    Student stud2;
    stud2.setName();
    stud2.setAge();
    stud2.setDegree();
    stud2.printData();
    return 0;
}

/* OUTPUT:
    Create a student: <name>, <age>, <degree>
    Enter name: student
    Enter age: 20
    Enter degree: bachelors
    Created a student: student, 20, bachelors
    Deleted student: student, 20, bachelors
*/
```

```
/*
W02 CLASSES Part B_01
    Create table with:
    Size three(3) that stores objects of the type Student.
*/

#include <iostream>
#include <vector>
//#include <string>
//---------------------student.h--------------------------------
class Student {
public:
    // Constructors
    Student();  //-Default constructor
    Student(std::string, int, std::string); //-Overloaded constructor
    ~Student(); //-Destructor

    //-Accessors (const = not modifiable)
    std::string getName() const;     //-return name
    int getAge() const;             //-return age
    std::string getDegree() const;   //-return degree

    //-Mutators (modifiable)
    void setName();     //-modify name
    void setAge();      //-modify age
    void setDegree();   //-modify degree

    // Print method
    void printData();

private:
    std::string Name;
    int Age;
    std::string Degree;
};
//-------------------------------------------------------

//-Default constructor (:: = 'scope resolution')
Student::Student() {
    Name = "John Doe";
    Age = -1;
    Degree = "Other degree";
}
//-Overloaded constructor
Student::Student(std::string n, int a, std::string d) {
    Name = n;
    Age = a;
    Degree = d;
}
//-Destructor
Student::~Student() {}

//-----------Get----------------------
std::string Student::getName() const {return Name;}
int Student::getAge() const {return Age;}
std::string Student::getDegree() const {return Degree;}
```

```cpp
//-----------Set----------------------
// Create a vector with new_students adress
void fillVec(std::vector<Student>& new_students){
    std::string n;
    int a;
    std::string d;

    std::cout<<"How many students? ";
    int size;
    std::cin>>size;

    for (int i=0; i<size; i++) { //Create as many students as in "int size"
        std::cout<<"Student("<<i+1<<") name: ";
        std::cin>>n;
        std::cout<<"Student("<<i+1<<") age: ";
        std::cin>>a;
        std::cout<<"Student("<<i+1<<") degree: ";
        std::cin>>d;

        // Store all new students in vector
        Student newStud(n,a,d);
        new_students.push_back(newStud);
    }
}

// Print method from vector with adress new_students
void printVec(const std::vector<Student>& new_students) {
    // Get number of students and print theese out
    for (int i=0; i<new_students.size(); i++) {
        std::cout   <<"Created a student: "<<new_students[i].getName()
                    <<", "<<new_students[i].getAge()
                    <<", "<<new_students[i].getDegree()<< std::endl;
    }
}

int main() {
    std::vector<Student> students;
    fillVec(students);
    printVec(students);
    return 0;
}

/* OUTPUT:
    How many students? 2
    Student(1) name: Student1
    Student(1) age: 20
    Student(1) degree: Bach
    Student(2) name: Student2
    Student(2) age: 21
    Student(2) degree: Bach2
    Created a student: Student1, 20, Bach
    Created a student: Student2, 21, Bach2
*/
```

```
/*
W02 CLASSES Part B_02
    Create table with:
    Size three(3) that stores objects of the type pointer to Student.
*/
#include <iostream>
#include <vector>
//#include <string>
//---------------------student.h---------------------------------
class Student {
public:
    //Constructors
    Student();  //-Default constructor
    Student(std::string, int, std::string); //-Overloaded constructor
    ~Student(); //-Destructor

    //-Accessors (const = not modifiable)
    std::string getName() const;     //-return name
    int getAge() const;             //-return age
    std::string getDegree() const;   //-return degree

    //-Mutators (modifiable)
    void setName();     //-modify name
    void setAge();      //-modify age
    void setDegree();   //-modify degree

    // Print method
    void printData();

private:
    std::string Name;
    int Age;
    std::string Degree;
};
//-----------------------------------------------------------

//-Default constructor (:: = 'scope resolution')
Student::Student() {
    Name = "John Doe";
    Age = -1;
    Degree = "Other degree";
}
//-Overloaded constructor
Student::Student(std::string n, int a, std::string d) {
    Name = n;
    Age = a;
    Degree = d;
}
//-Destructor
Student::~Student() {
}

//-----------Get----------------------
std::string Student::getName() const {return Name;}
int Student::getAge() const {return Age;}
std::string Student::getDegree() const {return Degree;}

//-----------Set----------------------
// Create a vector with new_students adress and point to it
void fillVec(std::vector<Student*>& new_Vec_students) {
    Student* p_students = NULL;
    std::string n;
    int a;
    std::string d;
    std::cout<<"How many students? ";
```

```
    int size;
    std::cin>>size;

    for (int i=0; i<size; i++) {
        std::cout<<"Student("<<i+1<<") name: ";
        std::cin>>n;
        std::cout<<"Student("<<i+1<<") age: ";
        std::cin>>a;
        std::cout<<"Student("<<i+1<<") degree: ";
        std::cin>>d;
        std::cout<<std::endl;
        p_students = new Student(n,a,d);
        new_Vec_students.push_back(p_students);
    }
}

// a->b =(*a).b, access member through pointer
void printVec(const std::vector<Student*> new_students) {
    for (int i=0; i<new_students.size(); i++) {
        std::cout<<"Created a student: "<<new_students[i]->getName()
        <<", "<<new_students[i]->getAge()
        <<", "<<new_students[i]->getDegree()<< std::endl;
        std::cout<<std::endl;
    }
}

// Clean up pointers after creation
static void cleanUP(std::vector<Student*>& students) {
    for (std::vector<Student*>::iterator pObj = students.begin(); pObj!=students.end(); ++pObj)
 {
        delete *pObj;
    }
    for (int i=0; i<students.size(); ++i) {
        std::cout<<"Deleting: "<<students[i]->getName()
        <<", "<<students[i]->getAge()
        <<", "<<students[i]->getDegree()<< std::endl;
    }
    students.clear();
}

int main() {
    std::vector<Student*> students;
    students.reserve(3);
    fillVec(students);
    printVec(students);
    cleanUP(students);
    return 0;
}

/* OUTPUT:
    How many students? 2
    Student(1) name: Stud
    Student(1) age: 20
    Student(1) degree: MS

    Student(2) name: Stud2
    Student(2) age: 21
    Student(2) degree: MS1

    Created a student: Stud, 20, MS
    Created a student: Stud2, 21, MS1
    terminate called after throwing an instance of 'std::logic_error'
        what():  basic_string::_M_construct null not valid
    Aborted (core dumped)
*/
```

```
/*
W02 CLASSES Part B_03
    Create a linked list of student objects.
    [Data|ptr]->[Data|ptr]->[Data|ptr]->Nullptr
*/
#include <iostream>
#include <vector>
//#include <string>
//---------------------student.h--------------------------------
class StudentNode {
public:
    // Constructors
    StudentNode();  //-Default constructor
    StudentNode(std::string, int, std::string); //-Overloaded constructor
    ~StudentNode(); //-Destructor

    std::string Name;
    int Age;
    std::string Degree;
    StudentNode* next;

    friend class StudentLink; //-
friend class: a class whose members have access to the private members of another class.
};
//-------------------------------------------------------
class StudentLink {
public:
    StudentLink();  //-Default constructor
    StudentLink(std::string, int, std::string); //-Overloaded constructor
    ~StudentLink(); //-Destructor

    void addData(std::string, int, std::string);
    void print();
private:
    StudentNode* head; // point to head
};

//-Default constructor (:: = 'scope resolution')
StudentNode::StudentNode() {
    Name = "John Doe";
    Age = -1;
    Degree = "Other degree";
}
//-Overloaded constructor
StudentNode::StudentNode(std::string n, int a, std::string d) {
    Name = n;
    Age = a;
    Degree = d;
}
//-Destructor
StudentNode::~StudentNode() {
    std::cout<<"Node deleted";
}

// this:
StudentLink::StudentLink() {
    this->head = NULL;
}

StudentLink::~StudentLink() {
    std::cout<<"\nList deleted\n";
}
void StudentLink::addData(std::string Name, int Age, std::string Degree) {
    StudentNode* Snode = new StudentNode();
    Snode->Name = Name;
```

```
    Snode->Age = Age;
    Snode->Degree = Degree;
    Snode->next = this->head;
    this->head = Snode;
}

void StudentLink::print() {
    StudentNode* head = this->head;
    while (head) {
        std::cout<<" -> ["<<head->Name<<", "
        <<head->Age<<", "<<head->Degree<<"]";
        head = head->next;
    }
}

//-----------Set----------------------
static void setValues(StudentLink *list) {
    std::string n;
    int a;
    std::string d;
    std::cout<<"How many students? ";
    int size;
    std::cin>>size;

    for (int i=0; i<size; ++i) {
        std::cout<<"Student("<<i+1<<") name: ";
        std::cin>>n;
        std::cout<<"Student("<<i+1<<") age: ";
        std::cin>>a;
        std::cout<<"Student("<<i+1<<") degree: ";
        std::cin>>d;
        std::cout<<std::endl;
        list->addData(n, a, d);
    }
}

int main() {
    StudentLink* list = new StudentLink();
    setValues(list);
    list->print();
    delete list;
    return 0;
}

/* OUTPUT:
    How many students? 2
    Student(1) name: Stud1
    Student(1) age: 20
    Student(1) degree: BS

    Student(2) name: Stud2
    Student(2) age: 21
    Student(2) degree: MS

    -> [Stud2, 21, MS] -> [Stud1, 20, BS]
    List deleted
*/
```

```
/*
W02 CLASSES Part C
    1. add a static data element to the Student class to store the number of students objects
       in existence.
    2. Create a function(not method) called Print() and that accepts a Student object as a
       'pass by value' transfer.
    3. Make sure that all constructor (including copy constructors) keeps track of number of
       objects and that the destructor does the same.
    4. Make a PrintNumberOfStudents() method that prints out number of student objects.
 */

#include <iostream>
#include <vector>
//#include <string>

// Curiously recurring template pattern
template <typename T>
struct counter {
    static int students_created;
    static int students_alive;
    counter() {
        ++students_created;
        ++students_alive;
    }
    counter(const counter&) {
        ++students_created;
        ++students_alive;
    }
protected: //Protected members are accessible in the class that defines them and in classes tha
t inherit from that class.
    ~counter(){
        --students_alive;
    }
};
template <typename T> int counter<T>::students_created(0);
template <typename T> int counter<T>::students_alive(0);

//-----------------------------------------------------
//class Student derives from a class template instantiation using Student itself as template ar
gument.
class Student : counter<Student> {
public:
    Student();
    Student(std::string, int, std::string);
    ~Student(){;}

    std::string getName() const;
    int getAge() const;
    std::string getDegree() const;
    void setName();
    void setAge();
    void setDegree();
    void printData();

private:
    std::string Name;
    int Age;
    std::string Degree;
};
//-----------------------------------------------------
Student::Student(std::string n, int a, std::string d) {
    Name = n;
    Age = a;
    Degree = d;
}
```

```cpp
//-----------Get----------------------
std::string Student::getName() const {return Name;}
int Student::getAge() const {return Age;}
std::string Student::getDegree() const {return Degree;}

//-----------Set----------------------
void fillVec(std::vector<Student>& new_students){
    std::string n;
    int a;
    std::string d;
    std::cout<<"How many students? ";
    int size;
    std::cin>>size;
    for (int i=0; i<size; i++) {
        std::cout<<"Student("<<i+1<<") name: ";
        std::cin>>n;
        std::cout<<"Student("<<i+1<<") age: ";
        std::cin>>a;
        std::cout<<"Student("<<i+1<<") degree: ";
        std::cin>>d;
        Student newStud(n,a,d);
        new_students.push_back(newStud);
    }
}

void printVec(const std::vector<Student>& new_students) {
    for (int i=0; i<new_students.size(); i++) {
        std::cout   <<"Created a student: "<<new_students[i].getName()
                    <<", "<<new_students[i].getAge()
                    <<", "<<new_students[i].getDegree()<< std::endl;
    }
}

void PrintNumberOfStudents(const std::vector<Student>& students){
    std::cout<<"\nNumber of student objects created = "<<counter<Student>::students_created<<st
d::endl;
    std::cout<<"\nNumber of student objects alive = "<<counter<Student>::students_alive<<std::e
ndl;
}

int main() {
    std::vector<Student> students;
    fillVec(students);
    printVec(students);
    PrintNumberOfStudents(students);
    students.clear(); //Removes all elements from the vector (which are destroyed), leaving the
 container with a size of 0.
    PrintNumberOfStudents(students);
    return 0;
}

/* OUTPUT:
    How many students? 2
    Student(1) name: Stud1
    Student(1) age: 20
    Student(1) degree: BS
    Student(2) name: Stud2
    Student(2) age: 21
    Student(2) degree: MS
    Created a student: Stud1, 20, BS
    Created a student: Stud2, 21, MS

    Number of student objects created = 5, Number of student objects alive = 2
    Number of student objects created = 5, Number of student objects alive = 0 */
```

```
/*
W03 INHERITANCE Part 01
    1. Create class Human that can store the data name og age:
    2. Create constructor
    3. get- and set-methods
    4. Print method
    5. Create two classes, Student and Professor, with the added data ects (for Student) and pa
pers (for Professors):
    6. Constructor calls the constructor in the base class and use that for initialising the cl
ass.
    7. get- and set-methods
    8.Print that also used the base class print method
 */
#include <iostream>
#include <string>

//-----------------------------------
class Human {
public:
    // Defalut constructor
    Human() {
        std::cout<<"Creating default Human"<<std::endl;
    }
    // Overloaded Constructor
    Human(std::string n, int a) {
        Name = n;
        Age = a;
        std::cout<<"Creating overloaded Human"<<std::endl;
    }

    // Get (accessors)
    std::string getName() const {return Name;}
    int getAge() const {return Age;}

    // Set (mutators)
    void setName(std::string n) {Name = n;}
    void setAge(int a) {Age = a;}

    // Print method
    virtual void printHuman() {
        std::cout<<"Name: "<<Name<<std::endl
        <<"Age: "<<Age<<std::endl;
    }

private: // Store these values privatly
    std::string Name;
    int Age;
};
//-----------------------------------
class Student : public Human {
public:
    Student(std::string Name, int Age, int e) : Human(Name, Age) {
        ECTS = e;
        std::cout<<"Creating overloaded Student"<<std::endl;
    }
    int getECTS() const {return ECTS;}
    void printStudent() {
        Human::printHuman(); //Print Human first then add Student ETCS
        std::cout<<"ECTS: "<<ECTS<<std::endl;
    }
private:
    int ECTS;
};
//-----------------------------------
class Professor : public Human {
```

```
public:
    Professor(std::string Name, int Age, int p) : Human(Name, Age) {
        Papers = p;
        std::cout<<"Creating overloaded Professor"<<std::endl;
    }
    int getPapers() const {return Papers;}
    void printProfessor() {
        Human::printHuman(); //Print Human first then add Professor Papers
        std::cout<<"Papers: "<<Papers<<std::endl;
    }
private:
    int Papers;
};
//-------------------------------------

int main() {
    Student stu("Student",20,300); // Set predefined Student name, age and ects
    stu.printStudent();

 //alt.
 //Student stu("Student",20,300);
 //Student* s_p = &stu;
 //s_p->printStudent();

    Professor pro("Prof.",30,50); // Set predefined Professor name, age and papers
    pro.printProfessor();
    return 0;
}

/* OUTPUT:
    Creating overloaded Human
    Creating overloaded Student
    Name: Student
    Age: 20
    ECTS: 300
    Creating overloaded Human
    Creating overloaded Professor
    Name: Prof.
    Age: 30
    Papers: 50
*/
```

```
/*
W03 INHERITANCE Part 02
    1. Create a class SecretAgent where the access to class Human is private.
    2.Add the data number (007 or similar) include this get and set methods,
        verify that these can be used in the main(), while the others can not.
 */
#include <iostream>
#include <string>

//-----------------------------------
class Human {
public:
    // Default constructor
    Human() {
        Name = "";
        Age = 0;
        std::cout<<"Creating default Human"<<std::endl;
    }
    // Overloaded constructor
    Human(std::string n, int a) {
        Name = n;
        Age = a;
        std::cout<<"Creating overloaded Human"<<std::endl;
    }

    // Get (accessors)
    std::string getName() const {return Name;}
    int getAge() const {return Age;}

    // Set (mutators)
    void setName() {
        std::string n;
        Name = n;
    }
    void setAge() {
        int a = 0;
        Age = a;
    }

    // Create a virtual void for other Classes to refer to
    virtual void printHuman() {
        std::cout<<"Name: "<<Name<<std::endl <<"Age: "<< Age << std::endl;
    }

private:
    std::string Name;
    int Age;
};

class SecretAgent : private Human { // SecretAgent class refers to private Human class
public:

    SecretAgent(){
    //Name = " "; //- private member of human
    //Age = 0;    //- private member of human
        agentNr = " ";
        std::cout<<"Creating default Agent"<<std::endl;
    }

    SecretAgent(std::string n, int a, std::string a_nr):Human(n,a) {
    //Name = n; //- private member of human
    //Age = a;  //- private member of human
        agentNr = a_nr;
        std::cout<<"Creating overloaded Agent"<<std::endl;
    }
```

```
    std::string getAgentNr () const {return agentNr;}
    void setAgentNr() {
        std::string a_nr = "007";
        agentNr = a_nr;
    }

    void printAgent() {
        Human::printHuman();
        std::cout<<"Agent: "<<agentNr<<std::endl;
    }

private:
    std::string agentNr;
};

//-------------------------------------
int main() {
    SecretAgent james;
    james.setAgentNr();
    james.printAgent();
    SecretAgent over("Alf",40,"008");
    over.printAgent();
    return 0;

    // only the agentNr will be printed.
}

/* OUTPUT:
    Creating default Human
    Creating default Agent
    Name:
    Age: 0
    Agent: 007
    Creating overloaded Human
    Creating overloaded Agent
    Name: Alf
    Age: 40
    Agent: 008
*/
```

```
/*
W04 TEMPLATES Task 1
    1. Compile and run the swap example
    2. Verify that that the program actually swaps the values.
    3. Can you swap the value of two objects of the class string or your home made CString?
        Discuss Why / Why not
*/

#include<iostream>

template<class X>
    void Swap( X &a, X &b ) // "&" is an adress
    {
        X temp;
        temp = a;
        a = b;
        b = temp;
    }

int main()
{
    int nI = 10, nJ = 2;
    char c1 = 'B', c2 = 'A';
    float f1 = 0.0, f2 = -1.0;
    std::string s1 = "S1", s2 = "S2";
    const char *Cs1 = "Cs1", *Cs2 = "Cs2"; // since it is a string it has to "*" point value to
 Csx object

    Swap( nI, nJ );
    Swap( c1, c2 );
    Swap( f1, f2 );
    Swap( s1, s2 );
    Swap( Cs1, Cs2);

    std::cout << "nI = "<< nI << std::endl;
    std::cout << "nJ = "<< nJ << std::endl;
    std::cout << "c1 = "<< c1 << std::endl;
    std::cout << "c2 = "<< c2 << std::endl;
    std::cout << "f1 = "<< f1 << std::endl;
    std::cout << "f2 = "<< f2 << std::endl;
    std::cout << "s1 = "<< s1 << std::endl;
    std::cout << "s2 = "<< s2 << std::endl;
    std::cout << "Cs1 = "<< Cs1 << std::endl;
    std::cout << "Cs2 = "<< Cs2 << std::endl;
}

/* OUTPUT:
    nI = 2, nJ = 10
    c1 = A, c2 = B
    f1 = -1, f2 = 0
    s1 = S2, s2 = S1
    Cs1 = Cs2, Cs2 = Cs1
*/
```

```
/*
W04 TEMPLATES task 2
    Implement these mathematical operations by using templates:
    answer = myAbs(x);
    answer = myAdd(x,y);
    answer = myPow(x,int);

1.  Demonstrate that your code works by printing:
    myAbs(-5) myAbs(-3.14)
    myAdd(5,-5) myAdd(2.71, -3.14) myAdd('1','G')
    myPow(3.14,10) myPow('c',2)

2.  What happens if we have two string-objects s1 "hei" and  s2 "hallo":
    myAbs(s1);
    myAdd(s1,s2);
    myPow(s2,2);
*/

#include <iostream>
#include <cmath>

template <typename T> // Template to show absolute value
T myAbs(T x){
    return (x*((x>0)-(x<0)));
    //return abs(x);
}

template <typename T> // Template to add values
T myAdd(T x, T y){
    return (x + y);
}

template <typename T> // Template to raise x to the power of y
T myPow(T x, int y){
    return pow(x, y);
}

int main(int argc, const char * argv[]) {
    std::string s1 = "hei";
    std::string s2 = "hallo";

    std::cout<<"myAbs: "<<myAbs(-5)<<std::endl;
    std::cout<<"myAdd: "<<myAdd(s1, s2)<<std::endl;
    std::cout<<"myPow: "<<myPow('c', 2)<<std::endl;

    return 0;
}

/* OUTPUT:
1. Demonstrate
    myAbs: -5
    myAdd: heihallo
    myPow: I

2. What happens if we have two string-objects s1 "hei" and  s2 "hallo":
    myAbs(s1); // doesnt work because of abs()
    myAdd(s1,s2); --> = heihallo
    myPow(s2,2); // doesnt work because of pow()

*/
```