Лабораторная работа №4

ПЕРЕГРУЗКА ОПЕРАЦИЙ

Цель. Получить практические навыки работы в среде BC++5.02 и создания Easy Win-программы. Получить практические навыки создания абстрактных типов данных и перегрузки операций в языке C++.

Основное содержание работы

Определить и реализовать класс - абстрактный тип данных. Определить и реализовать операции над данными этого класса. Написать и выполнить EasyWin-программу полного тестирования этого класса.

Краткие теоретические сведения Абстрактный тип данных (АТД)

АТД - тип данных, определяемый только через операции, которые могут выполняться над соответствующими объектами безотносительно к способу представления этих объектов.

АТД включает в себя абстракцию как через параметризацию, так и через спецификацию. **Абстракция через параметризацию** может быть осуществлена так же, как и для процедур (функций); использованием параметров там, где это имеет смысл. **Абстракция через спецификацию** достигается за счет того, что операции представляются как часть типа.

Для реализации АТД необходимо, во-первых, выбрать представление памяти для объектов и, во-вторых, реализовать операции в терминах выбранного представления.

Примером абстрактного типа данных является класс в языке C++.

Перегрузка операций

Возможность использовать знаки стандартных операций для записи выражений как для встроенных, так и для АТД.

В языке C++ для перегрузки операций используется ключевое слово **operator**, с помощью которого определяется специальная операция-функция (operator function).

Формат операции-функции:

типвозврзначения operator знакоперации (специфпараметров)

{операторытелафункции}

Перегрузка унарных операций

• Любая унарная операция © может быть определена двумя способами: либо как компонентная функция без параметров, либо как глобальная (возможно дружественная) функция с одним параметром. В первом случае выражение © Z означает вызов Z.operator © (), во втором - вызов operator ©(Z).

• Унарные операции, перегружаемые в рамках определенного класса, могут перегружаться только через нестатическую компонентную функцию без параметров. Вызываемый объект класса автоматически воспринимается как

операнд.

• Унарные операции, перегружаемые вне области класса (как глобальные функции), должны иметь один параметр типа класса. Передаваемый через этот параметр объект воспринимается как операнд.

Синтаксис:

а) в первом случае (описание в области класса):

типвозврзначения operator знакоперации

б) во втором случае (описание вне области класса):

типвозврзначения operator знакоперации(идентификатортипа)

Примеры.

1) class person { int age;

public:

void operator++(){ ++age;}

}; void main() {class person jon;

++jon;}

Перегрузка бинарных операций

• Любая бинарная операция © может быть определена двумя способами: либо как компонентная функция с одним параметром, либо как глобальная (возможно дружественная) функция с двумя параметрами. В первом случае функция означает вызов **x.operator®(y)**, во втором – вызов **operator Ф(x,y).**

• Операции, перегружаемые внутри класса, могут перегружаться только нестатическими компонентными функциями с параметрами. Вызываемый объект класса автоматически воспринимается в качестве первого операнда.

• Операции, перегружаемые вне области класса, должны иметь два операнда, один из которых должен иметь тип класса.

Примеры.

1) class person{...};

class adresbook

{ // содержит в качестве компонентных данных множество объектов типа //person, представляемых как динамический массив, список или дерево

public:

person& operator[](int); //доступ к i-му объекту };

person& adresbook : : operator[](int i){. . .} void main()

{class adresbook persons; class person record;

record = persons [3]; }

2) class person{…}; class adresbook

{ // содержит в качестве компонентных данных множество объектов типа//person, представляемых как динамический массив, список или дерево

… public:

friend person& operator[](const adresbook&,int);//доступ к i-му объекту };

person& operator[](const adresbook& ob ,int i){. . .} void main() {class adresbook persons;

class person record;

record = persons [3];}

Перегрузка операции присваивания Операция отличается тремя особенностями:

- операция не наследуется;
- операция определена по умолчанию для каждого класса в качестве операции поразрядного копирования объекта, стоящего справа от знака операции, в объект, стоящий слева.
- операция может перегружаться только в области определения класса. Это гарантирует, что первым операндом всегда будет леводопустимое выражение.

Формат перегруженной операции присваивания:

имя_классас& *орегМог=(нмя_класса&);*

Отметим две важные особенности функции *орегапог=*. Во-первых, в ней используется параметр-ссылка. Это необходимо для предотвращения создания копии объекта, передаваемого через параметр по значению. В случаи создания копии, она удаляется вызовом деструктора при завершении работы функции. Но деструктор освобождает распределенную память, еще необходимую объекту, который является аргументом. Параметр-ссылка помогает решить эту проблему.

Во-вторых, функция operator=() возвращает не объект, а ссылку на него. Смысл этого тот же, что и при использовании параметра-ссылки. Функция возвращает временный объект, который удаляется после завершения ее работы. Это означает, что для временной переменной будет вызван деструктор, который освобождает распределенную память. Но она необходима для присваивания значения объекту. Поэтому, чтобы избежать создания временного объекта, в качестве возвращаемого значения используется ссылка.

Создание приложений в Borland C++ x.0x

* Проекты и узлы.

Проект - это файл, содержащий все необходимое для построения конечного продукта: установленные параметры, информацию о целевой среде и о входных файлах. Файл проекта имеет расширение **ide**.

Термин “**узел**” применяется для обозначения различных объектов, находящихся в окне проекта. Каждый узел может зависеть от одного или

большего количества узлов. Это означает, что до обработки такого узла, узлы, от которых он зависит, должны быть успешно обработаны. Самый верхний узел в иерархии узлов называется целью. Можно внести больше, чем одну целевую программу в файл **.ide**. Рекомендуется сохранять в одном проекте файлы, связанные по смыслу.

* Программа из одного модуля.

1. Если исходный файл существует, откройте его, выбрав меню **File|Open.** В противном случае **-** меню **File|New**, команда **Text Edit.** Введите исходный код в новое окно редактора и сохраните в файле, выбрав меню **File|Save.**

2. Активизируйте локальное меню редактора, нажав правую кнопку мыши в середине окна(или **Alt+F10**) и выберите опцию **Target Expert.** Появится окно диалога **Target Expert.**

3. Выберите для вашей программы подходящие параметры и нажмите **Ок**. Например, можно построить программу как стандартное приложение **DOC**, как приложение **EasyWin** для Windows 3.x (16-разрядное) или типа **Concole** для Win32. Рекомендуется строить **EasyWin**-программу.

4. В меню **Debug** (отладка) выберите **Run** (выполнить). Программа будет откомпилирована, отредактирована и выполнена.

5. Для ускорения повторной компиляция в случае обнаружения и исправления ошибок рекомендуется установить режим прекомпиляции и сохранения откомпилированного кода заголовочных файлов. Для этого в меню **Options|Project|Compiler** выберите опцию **Precompiled Headers** и установите флажок **Generate and use.**

* Проекты и многомодульные программы.

Если для создания программы используется несколько исходных модулей, вы должны организовать проект. Для организации проекта:

1. В меню **File|New** выберите **Project.** Появится окно **New Target.** Это расширенный вариант окна **Target Expert.**

2. В окне укажите имя проекта (**Project Path and Name**), имя и тип входного модуля (**Target Name**) и требуемые спецификации библиотек C++.

3. Выберите кнопку **Advanced**, чтобы уточнить проект. Появится окно **Advanced Options.**

4. Если программа не применяет управление ресурсами и не включает в себя файл **.def**, выключите селекторы **.rc** и **.def.**

5. Закройте окна **Advanced Options** и **New Target** (кнопкой **Ok**).

6. В появившемся окне **Project** с помощью правой кнопки мыши укажите узел, чтобы активизировать его локальное меню. Чтобы включит в проект новые модули, выберите в меню опцию **Add Node.** Появится окно **Add to Proect List**, которое позволит вам найти и указать те файлы, которые нужно включить в

проект.

7. После того, как вы подключили новые узлы, с помощью правой кнопки мыши укажите на целевой узел. Теперь, чтобы построить программу, выберите опцию **Make Node.**

8. После того, как вы создали и сохранили проект, вы сможете в дальнейшем загружать его в **IDE** командой **Open Project** в меню **Project.** При этом загружаются все связанные с ним файлы.

*Построение нескольких целевых модулей.

1. В меню **Project** выберите **New Target.** Появится окно диалога **Add Target.**

2. Введите имя целевого модуля, установите тип цели (**Standard**) и нажмите **Ok.** Появится окно **New Target.**

3. Установите в окне **New Target** требуемые параметры.

4. Если необходимо вновь созданный целевой узел сделать подузлом другого узла, то с помощью левой кнопки мыши “перенесите” узел и “положите” его на узел, подузлом которого он будет.

5. Вызовите правой кнопкой мыши локальное меню целевого узла и выберите **Build Node**, чтобы построить программу.

Что такое EasyWin-программа?

Интегрированная среда разработки(**IDE**) BC++5.02 дает возможность создавать специальный вид программ, называемых EasyWin-программами, которые выполняются в простом окне, напоминающем окно Windows. Это окно содержит стандартные кнопки и полосы скроллера. Текст в окне выводится в кодировке Windows, что не создает проблем с кириллицей.

*Порядок создания EasyWin-программы.

1. Загрузите IDE BC++5.02.

2. Выберите меню **File.**

3. Выберите команду **New/Project,** вызывающую окно **New Target.**

4. Введите маршрут и имя файла проекта **.ide** в поле ввода в верхней части окна (поле **Project Path and Name**) . Введенное имя будет повторено в поле ввода имени цели (поле **Target Name***)*, тем самым имя программы будет соответствовать имени проекта. Кнопка **Brows** служит для выбора каталога, содержащего файлы проекта.

5. Выберите **EasyWin[.exe]** в списке **Target Type**.

6. Щелкните на кнопке **Advanced,** чтобы вызвать диалоговое окно **Advanced Options**. Выберите в нем переключатель, помеченный как **.cpp Node**. Этот выбор заставит IDE вставлять **.cpp-**узлы. Это диалоговое окно дает также возможность добавлять или удалять модули **.rc** и **.def**. Так как реально они нужны только при создании Windows-приложений, а не EasyWin, удостоверьтесь, что эти две кнопки не выбраны. Закройте окно (**Ok**).

7. Нажмите **Ok**, чтобы создать новый файл проекта.

8. IDE выведет окно проекта **Project**, в котором перечислены узлы различных программ. Когда вы создаете новый файл проекта, окно **Project** будет содержать только один узел.

9. Узлы программы Easy Win содержат только один файл **- .cpp** с текстом программы. Дважды щелкните на файле **-.cpp**, для того чтобы начать редактирование этого файла.

10. Введите исходный текст программы.

11. Откомпилируйте программу (**F9**).

12. Исправьте ошибки и повторите компиляцию.

13. Повторяйте пункт 12, пока компиляция не будет выполняться без ошибок.

14. Выполните программу (**Ctrl+F9**).

Порядок выполнения работы

1. Выбрать класс АТД в соответствии с вариантом.

2. Определить и реализовать в классе конструкторы, деструктор, функции Input (ввод с клавиатуры) и Print (вывод на экран), перегрузить операцию присваивания.

3. Написать программу тестирования класса и выполнить тестирование.

4. Дополнить определение класса заданными перегруженными операциями (в соответствии с вариантом).

5. Реализовать эти операции. Выполнить тестирование.

Методические указания

1.Класс АТД реализовать как динамический массив. Для этого определение класса должно иметь следующие поля:

- указатель на начало массива;
- максимальный размер массива;
- текущий размер массива.

2. Конструкторы класса размещают массив в памяти и устанавливают его максимальный и текущий размер. Для задания максимального массива использовать константу, определяемую вне класса.

3. Чтобы у вас не возникало проблем, аккуратно работайте с константными объектами. Например:

*конструктор копирования следует определить так: MyClass (**const** MyClass& ob);

*операцию присваивания перегрузить так: MyClass& operator = (**const** MyClass& ob);

4. Для удобства реализации операций-функций реализовать в классе **private(protected**)-функции, работающие непосредственно с реализацией класса. Например, для класса **множество** это могут быть следующие функции:

- включить элемент в множество;
- найти элемент и возвратить его индекс;
- удалить элемент;
- определить, принадлежит ли элемент множеству.

Указанные функции используются в реализации общедоступных функций-операций (operator).

Задания для лабораторной работы

1. АТД - множество с элементами типа **char.** Дополнительно перегрузить следующие операции:

+ - добавить элемент в множество(типа char + set);

+ - объединение множеств;

== - проверка множеств на равенство.

2. АТД - множество с элементами типа **char.** Дополнительно перегрузить следующие операции:

- - удалить элемент из множества (типа set-char);

* - пересечение множеств;

< - сравнение множеств.

3. АТД - множество с элементами типа **char.** Дополнительно перегрузить следующие операции:

- - удалить элемент из множества (типа set-char);

> - проверка на подмножество;

!= - проверка множеств на неравенство.

4. АТД - множество с элементами типа **char.** Дополнительно перегрузить следующие операции:

+ - добавить элемент в множество (типа set+char); * - пересечение множеств; int()- мощность множества.

5. АТД - множество с элементами типа **char.** Дополнительно перегрузить следующие операции:

() - конструктор множества (в стиле конструктора Паскаля); + - объединение множеств; <= - сравнение множеств .

6. АТД - множество с элементами типа **char.** Дополнительно перегрузить следующие операции:

> - проверка на принадлежность(спаг in set Паскаля);

* - пересечение множеств;

< - проверка на подмножество.

7. АТД - однонаправленный список с элементами типа **char.** Дополнительно перегрузить следующие операции:

+ - объединить списки (list+list);

— удалить элемент из начала (типа —list);

== - проверка на равенство.

8. АТД - однонаправленный список с элементами типа **char.** Дополнительно перегрузить следующие операции:

+ - добавить элемент в начало(спаг+Н81);

— удалить элемент из начала(типа -list);

== - проверка на равенство.

9. АТД - однонаправленный список с элементами типа **char.** Дополнительно перегрузить следующие операции:

+ - добавить элемент в конец (list+char);

---- удалить элемент из конца (типа list--);

!= - проверка на неравенство.

10. АТД - однонаправленный список с элементами типа **char.** Дополнительно перегрузить следующие операции:

[] - доступ к элементу в заданной позиции, например: hit i; char c;

list L;

c=L[i];

+ - объединить два списка;

== - проверка на равенство.

11. АДТ - однонаправленный список с элементами типа **char.** Дополнительно перегрузить следующие операции:

[] - доступ к элементу в заданной позиции, например:

hit i; char c;

list L;

c=L[i];

+ - объединить два списка;

!= - проверка на неравенство.

12. АДТ - однонаправленный список с элементами типа **char.** Дополнительно перегрузить следующие операции:

() - удалить элемент в заданной позиции, например :

hit i;

list L;

L[i];

() - добавить элемент в заданную позицию, например :

hit i; char c;

list L;

L[c,i];

!= - проверка на неравенство.

13. АДТ - стек. Дополнительно перегрузить следующие операции:

+ - добавить элемент в стек;

— извлечь элемент из стека;

bool()- проверка, пустой ли стек.

14. АДТ - очередь. Дополнительно перегрузить следующие операции:

+ - добавить элемент;

— извлечь элемент;

bool() - проверка, пустая ли очередь.

15. АДТ - одномерный массив (вектор) вещественных чисел. Дополнительно перегрузить следующие операции:

+ - сложение векторов (a[i]+b[i] для всех i);

[] - доступ по индексу;

+ - добавить число к вектору (double+vector).

16. АТД - одномерный массив (вектор) вещественных чисел. Дополнительно перегрузить следующие операции:

- - вычитание векторов (a[i]-b[i] для всех i); [] - доступ по индексу;

- - вычесть из вектора число (vector-double).

17. АТД - одномерный массив (вектор) вещественных чисел. Дополнительно перегрузить следующие операции:

* - умножение векторов (a[i]*b[i] для всех i); [] - доступ по индексу;

* - умножить вектор на число (vector*double).

18. АТД - одномерный массив (вектор) вещественных чисел. Дополнительно перегрузить следующие операции:

int() - размер вектора;

() - установить новый размер;

- - вычесть из вектора число (vector-double);

[] - доступ по индексу;

19. АТД - одномерный массив (вектор) вещественных чисел. Дополнительно перегрузить следующие операции:

= - присвоить всем элементам вектора значение (vector=double);

[] - доступ по индексу;

== - проверка на равенство;

!= - проверка на неравенство;

20. АТД - двухмерный массив (матрица) вещественных чисел. Дополнительно перегрузить следующие операции:

() - доступ по индексу;

* - умножение матриц;

* - умножение матрицы на число;

* - умножение числа на матрицу.