Программирование на ЯВУ (3 семестр)

Преподаватель: Носовицкий Вадим Борисович
Записано: Кузенков Артём, ИДБ-22-03

———————————————————————————————————————

1. Специальные функции-элементы класса

13.09.23

К специальным функциям-элементам класса кроме конструктора и деструктора относятся конструктор копии, операция присваивания, конструктор преобразования.
Кроме этого к таким функциям принято относить функции, обеспечивающие преобразование объектов класса в другие типы.

Конструктор копии — это конструктор специального вида, который имеет ровно 1 параметр — ссылку на объект этого же класса. Если конструктор копии не создавался, то генерируется конструктор копии по умолчанию, создающий буквальную копию объекта.

```
class salieva {
        int a, b;
        public:
        salieva (int aa, int bb) : a(aa), b(bb) { } // список инициализации
        salieva (salieva & proval) // Первая — конструктор, 2 — тип пер-ой
                { a = proval.a * 5;
                b = proval.b - 5; }
        salieva & operator = (const salieva &); // прототип
        salieva (int starosta) { // для myagkov
                a = starosta + 5;
                b = starosta - 6; }
        operator int ( ) {
                return 2 * a + 3 * b - 11; } // a и b от myagkov
        };

salieva & salieva :: operator = (const salieva & elkin) {
        a = elkin.a;
        b = elkin.b - 666;
        return *this; } // указатель на текущий объект

void main (void) {
        salieva diskretka (25,25), frezer (30,30);
        salieva evgenich = diskretka; // или (diskretka)
        frezer = diskretka;
        salieva myagkov = 54;

        int zaraza;
        zaraza = (int) myagkov; // явное преобразование типа
        }
```

Обозначение ссылки в заголовке, = — вызов конструктора копии.

Операция присваивания — это функция с именем operator =, которая принимает в качестве своего единственного параметра ссылку на объект данного класса. Эта функция вызывается тогда, когда одному уже существующему объекту класса присваивается значение другого существующего объекта класса. Если операцию присваивания не определить явно, то выполняется автоматическое присваивание, создающее точную копию объекта.

Среди конструкторов класса отдельную группу составляют конструкторы преобразования, к которым относятся все конструкторы с одним параметром.

При необходимости автор класса может запретить вызов конструктора преобразования с использованием знака =. Для такого запрета в объявлении конструктора должно быть добавлено ключевое слово explicit.

В классе можно определять элементы-функции, которые будут обеспечивать явное преобразование типа данного класса в другие типы.

Заголовки таких функций имеют вид

```
operator <имя_типа> ( ) { … };
```

Такие функции не могут иметь параметров, и для них не определяется тип возвращаемого значения, так как подразумевается тип, стоящий за ключевым словом operator в имени функции.

2. Понятие о дружественности

20.09.23

Язык C++ позволяет разрешить доступ к элементам класса, объявленым с меткой private или protected, функциям других классов или даже функциям, не относящимся ни к каким классам. Это делается при помощи ключевого слова friend.

```
class proverka { int a, b;
        friend class rabota; // объявлен другом класс
        friend void otchislenie :: idb2204 (int); // объявлена функция из класса
        friend void print (const proverka&); // (1*) объявлена функция
        public: ~proverka ( );
        };
```

Строки, начинающиеся с ключевого слова friend, можно размещать в теле класса, не глядя на метки доступа. Дружественность не взаимна (если класс A объявил класс B другом, то это не означает, что A является дружественным для B).

Дружественность не передаётся по наследству (если класс A объявил класс B другом, то это не означает, что наследники A автоматически считают B другом, а также A — наследников B).

```
void print (const proverka & diskretka) {
        cout << diskretka.a << “#” << diskretka.b; } // не работает при удалении 1*
```

3. Наследование классов

Язык C++ позволяет классу наследовать элементы данных и методы от одного или нескольких другим классах. Новый класс называется производным классом. Класс, элементы которого наследуются, называется базовым классом. В свою очередь производный класс может служить базовым для другого класса.

27.09.23

За счёт наследования реализуется один из базовых принципов ООП — инскапсуляция. Смысл инкапсуляции состоит в том, что некоторые свойства и поведение объектов объединяются в одно целое, причём это целое закрыто для внешней обработки.

Синтаксис объявления класса наследника имеет следующий вид:

```
class [имя_наследника] : [ключ_доступа] [имя_базового_класса] { … };
```

Класс-наследник может только добавлять данные. Функции можно добавить или переопределять.

Ключ доступа, стоящий в заголовке определения класса наследника может быть public, private или protected. По умолчанию принимается private при использовании ключевого слова class, и public при использовании ключевого слова struct.

Этот ключ доступа влияет на уровень доступа к элементам базового класса через объекты производного класса. Доступ определяется в соответствии с более жёстким ключом доступа из пары: ключ доступа в заголовке класса наследника и ключ доступа в теле базового класса.

При наследовании не происходит автоматического наследования конструктора.

```
class prolet : public proverka {
        char * stroka;
        public:
        prolet (char *, int, int);
        friend void print (const proverka &);
        void print1 ( ) {
                proverka zaval (a, b);
                print (zaval);
                printf (“%s”, stroka); }

prolet::prolet (char * str, int a1, int b1): proverka (a1, b1) {
        strcpy (stroka, str); }

void main (void) {
        prolet poezdka (“Psel von”, 25, 25);
        poezdka.print1 ( ); }
```

Этот пример будет работать только в случае, если в базовом классе элементы данных будут объявлены как protected, а не как private.

C++ допускает так называемое сложное наследование, то есть одновременное наследование от двух и более базовых классов.

```
class leonid: public salieva, protected elkin { … };
```

При этом в иерархии классов может оказаться так, что класс косвенно унаследует несколько элементов данных от одного и того же базового класса.

4. Виртуальные функции

Среди особенностей ООП выделяют полиморфизм — способность различной реакции на внешне идентичный запрос в зависимости от того, от кого этот запрос исходит. Механизм виртуальной функции C++ как раз является примером проявления полиморфизма.

04.10.23

```
#include <iostream.h>
class IDB2201 {
        public:
        double tryp (float x) {return 5*x;}
        double giena (float x) {return 2+tryp;}
}
class IDB2204 : public IDB2201 {
        public:
        double tryp (float x) {return 5*10*x;}
}

void main (void) {
        IDB2204 starosta; // может быть и IDB2201, тут не важно
        cout << starosta.giena (2) << endl; }
```

При отсутствии в определении функции ключевого слова virtual в данном примере полиморфной реакции не наблюдается. Если же добавить это ключевое слово перед типом возвращаемого значения в базовом классе (также можно прописать это слово и во всех наследниках), то полиморфная реакция будет наблюдаться.

Виртуальность будет сохраняться до тех пор, пока в цепочке не изменяется список параметров. При этом виртуальные функции могут замещать и невиртуальные функции, но такая особенность не приветствуется разработчиками. Более того, принято объявлять так называемые чистые виртуальные функции, у которых нет тела.

```
virtual double tryp (float x) = 0;
```

Такие функции создаются в базовом классе, чтобы затем переопределяться в классах, производных от него. Класс, объявляющий одну или несколько чисто виртуальных функций, является абстрактным базовым классом. У такого класса нельзя создать представителя, а нужен он только в качестве базового для дальнейшего порождения класса, которое будут реализовывать его чисто виртуальные функции.

5. Динамическое распределение памяти в C++

В C++ для управления динамическим распределением памяти используются операции new и delete для простых переменных, а также new [ ] и delete [ ] для массивов. Наряду с этими механизмами сохранена возможность вызова стандартных функций языка C — malloc ( ), calloc ( ), free ( ) и т. д.

```
int * idb2203;
idb2203 = new int;
…
delete idb2203;
idb2203 = new int [25];
…
delete [ ] idb2203;
```

При динамическом распределении памяти можно сразу и инициализировать переменные.

```
int * elkin = new int (25);
cout << * elkin << endl;
```

Если операция new вызывается для представителя какого-либо класса, то одновременно вызывается и конструктор.

```
proverka * dvoechnik;
dvoechnik = new proverka (25, 25);
```

Для переменных, для которых память выделяется подобным образом, конструктор не занимается выделением памяти для объектов, а деструктор, а деструктор е не высвобождает. Этим занимаются операции new и delete.

Класс может определять свои собственные операции new и delete.

```
void * proverka :: operator new [ ] (…) { }
```

6. Стандартная схема проекта

Заказчик — формирует бизнес-требования (пишутся в терминах решаемой проблемы, часто их пишет не заказчик, а бизнес-аналитик).

Аналитик — формирует техническое задание. В нём структуры хранения данных, алгоритмы их обработки, описание интерфейсов.

11.10.23

В случае, когда речь идёт о разработке новой подсистемы, ТЗ попадает к архитектору, который осуществляет выбор программного обеспечения (среды) для реализации ТЗ.

После выбора среды (сред) реализации задача попадает к программисту (группе программистов). После завершения работы над программой и элементарной проверки программа попадает к тестировщику.

Как правило, тестировщик запускает тесты, составленные аналитиком, и если программа на тестовых примерах работает некорректно, то она возвращается в разработку.

После завершения тестирования программа документируется (инструкция пользователя, инструкция администратора и т. д.)

Затем наступает этап внедрения (сначала тестовая эксплуатация, затем полноценная эксплуатация).

Порядок работ, который описан выше, может уменьшаться, часть этапов при некоторых стратегиях разработки может отсутствовать. Полная схема иногда носит название каскадной разработки.

Наравне с каскадной разработкой используются различные варианты гибких методологий (обобщённое название — agile). Среди этих методологий особо выделяется scrum.

7. Понятие о восходящем подходе

Как правило, проектирование программного продукта производится сверху вниз, а вот программирование часто выполняется снизу вверх.

При таком подходе вначале изготавливаются модули самого низкого уровня, затем эти модули тестируются и происходит переход на уровень вверх.

Основная трудность при восходящем подходе состоит в том, что каждый модуль может правильно работать при выполнении тестов, рассчитанных на проверку данного модуля, но терять свою работоспособность при включении в общий программный проект. Это может быть связано с изменением ТЗ в ходе работы над проектом, с разной интерпретацией разными программистами тех или иных частей ТЗ, с особенностью реальных данных и т. д.

Эти трудности обычно концентрируются на последней фазе работы над программным проектом в условиях острой нехватки времени.

При возникновении подобных ситуаций применяются следующие решения:
1. Вносятся изменения в уже написанные модули
2. Перепрограммируется большая часть модулей
3. Вносятся изменения в проект, чтобы сделать его соответствующим готовой программе

18.10.23

8. Нисходящая разработка программ

При нисходящей разработке, и проектирование, и программирование ведутся сверху вниз. Самым первым разрабатывается модуль самого верхнего уровня. Вместо программ нижнего уровня (до тех пор, пока они не разработаны) используются «программные заглушки», которые нужны только для того, чтобы позволить программе верхнего уровня быть выполненной и проверенной.

В рамках нисходящего потока возможно несколько вариантов последовательного программирования модулей:
1. Иерархический
2. Операционный
3. Смешанный

При иерархическом подходе порядок программирования и тестирования модулей определяется их расположением в схеме иерархии. Сначала программируются и тестируются все модули одного уровня, после чего происходит переход на уровень ниже. В конце проекта реализуются очень много модулей, что может затруднить программирование и тестирование.

При операционном подходе модули разрабатываются в порядке их выполнения в готовой программе (если таковая существует). Одним из преимуществ этого подхода является возможность использования для тестирования модулей, обрабатывающих данные, результатом работы ранее созданных модулей, порождающих данные. Одним из недостатков является «выстраивание» разработчиков в очередь в ожидании написания тех или иных модулей.

Лучше всего использовать комбинацию иерархического и операционного подходов — смешанных подход. При определении порядка разработки модулей обычно учитываются следующие факторы:
1. К каждому вновь разрабатываемому модулю должен существовать путь управления
2. Должны быть доступны данные, получаемые от модулей или от «заглушек»
3. Доступность ресурсов (программистов)
4. Необходимо как можно раньше сделать модули, которые можно проверить и показать заказчику
5. При прочих равных следует начинать с более сложных модулей
6. Модули, обрабатывающие правильные данные, должны программировать раньше модулей, имеющих дело с неправильными данными

9. Понятие о модулях

Модуль — это последовательность логически связанных фрагментов, оформленных как отдельная часть программы.

Основными преимуществами модульного программирования являются:
1. Возможность разделения работы между несколькими программистами
2. Возможность создания библиотек модулей для их дальнейшего использования
3. Увеличение числа контрольных точек для наблюдения за ходом выполнения проекта
4. Облегчение тестирования

27.10.23

Модуль имеет 3 основных атрибута: он выполняет одну или несколько функций, обладает некоторой логикой и используется в одном или нескольких контекстах.

Функция — это внешнее описание модуля. Описывается, что делает модуль, но не как это делается. Логика описывает внутренний алгоритм модуля, то есть, как он выполняет свою функцию. Контекст описывает конкретное применение модуля.

Таким образом, очевидно, что любая программа должна разбиваться на модули, которые должны быть максимально независимы. Добиться этой независимости можно с помощью двух методов оптимизации: усилением внутренней связи в каждом модуле и ослаблении связи между модулями.

Прочность модулей — это мера его внутренних связей. Нужно стремиться реализовать отдельные функции отдельными модулями. Это называется высокой прочностью модуля. И нужно предельно ослабить связь между модулями по данным (слабое сцепление модулей).

1. Модуль, прочный по совпадению — модуль, между элементами которого нет осмысленных связей. Такие модули возникают при разбиении на модули уже после написания программы
2. Модуль, прочный по логике — это модуль, который при каждом вызове выполняет выбранную функцию из набора связанных с ним
3. Модуль, прочный по классу — это модуль, который последовательно выполняет набор связанных с ним функций, которые непосредственно относятся к процедуре решения задач
4. Информационно прочный модуль — модуль, который выполняет несколько функций. Причем все они работают с одной и той же структурой данных, и каждая представляется собственным входом
5. Функционально прочный модуль — модуль, выполняющий одну определённую функцию

Сцепление модулей — мера взаимодействия модулей по данным, которая характеризуется как способом передачи данных, так и свойствами самих этих данных.
1. Два модуля сцеплены по общей области если они ссылаются на одну и ту же глобальную структуру данных
2. Два модуля сцеплены по внешним данным если они ссылаются на один и тот же глобальный элемент данных
3. Два модуля сцеплены по формату, если они ссылаются на одну и ту же не глобальную структуру данных
4. Два модуля сцеплены по данным, если вызывает другой и все входные и выходные параметры — простые элементы данных

В дополнение к прочности и сцеплению есть и другие характеристики, влияющие на независимость модулей: размер (считается, что не должно быть более 200 строк) или предсказуемость модуля (модуль не должен хранить в следы своих состояний при последовательных вызовах).

08.11.23

При проектировании структуры программы имеется несколько стратегий разбиения на модули. Разбиение «исток — преобразование — сток» предполагает деление задачи на функции, занимающиеся получением данных, изменением их формы и затем доставкой их в некоторую точку вне задачи.

Функциональное разбиение — это деление задачи на функции, выполняющие конкретные преобразования данных.

10. Отладка программ

Отладка — это процесс действий, которые необходимо выполнить, когда точно известно, что программа не работает.

Тестирование — это последовательные и систематические попытки добиться ошибки от программы, которая считается работающей.

Для отладки большинство современных сред разработки включают свой состав специальные программы — отладчики, предоставляющие программисту широкий спектр возможностей. При этом многие программисты предпочитают использовать метод отладочной печати.

Типовыми советами при отладке программ могут быть:
1. Необходимо проверить соответствие форматов при вводе-выводе
2. Необходимо убедиться в отсутствии неинициализированных переменных
3. При поиске ошибок полезно просматривать стек вызова функций
4. При обнаружении ошибки необходимо проверить отсутствие этой или подобной ошибки в других частях программного кода

Считается, что при невозможности обнаружения ошибки действенным методом является попытка объяснить свой код кому-нибудь ещё.

Для полноценной отладки необходимо сделать ошибку воспроизводимой. При поиске ошибок часто помогает просмотр журнального файла (log). Также следует регистрировать любым способом все действия, которые производятся над программой в процессе отладки.

Если при добавлении отладочного кода (при просмотре программы по шагам) ошибка перестаёт проявляться, то проблема скорее всего связана с распределением памяти.

Иногда проблемы возникают из-за разных версий ПО, установленного на разных компьютерах. Если программа не работает у одного пользователя и работает у другого, то проблема часто бывает связана с правами доступа пользователей или с настройками на их компьютерах.

При использовании чужих модулей в качестве чёрного ящика часто выясняется, что модули, которые считались универсальными, не могут быть использованы при некоторых условиях.

22.11.23

11. Тестирование программ на ЯВУ

Темы: Граница между программистом и тестировщиков, Систематическое тестироваие как постоянное тестирование по ходу проекта, Нагрузочное тестирование, Стрессовое тестирование, Автоматизация тестирования

Простейшие свойства программ принято контролировать ещё на этапе создания. Одни из важнейших методов тестирования является тестирование граничных условий. Если при экстремальных условиях код работает корректно, то с высокой вероятностью он будет корректно работать повсюду.

```
int i, max;
char s[max];
for (i=0; s[i]=getchar( )!='\n' && i<max-1; i++);
s[i]='\0';
```

В примере не обрабатываются строки, не содержащие '\n'.

```
int provalishe (int n) {
        int f;
        f = 1;
        while (n--)
                f *= n;
        return f; }
```

Кроме граничных условий надо проверить несколько регулярных ситуаций, после чего программу можно передавать на тестирование другому специалисту.

Систематическое тестирование — это, с одной стороны, постоянное тестирование по ходу разработки вновь включаемых в состав проекта модулей, а с другой стороны, под этим термином понимается разработка системы тестов, обеспечивающих полную проверку алгоритма.

Даже самые простые программы требуют большого количества тестов.

ДЗ: рассмотрим набор тестов для полной проверки программы, входящей 3 целых числа и печатающей сообщение о том, является ли треугольник разносторонним, равнобедренным или равносторонним.

29.11.23

Ответ: все возможные комбинации 0 и 1 кроме 1 1 1, комбинации 0 и -1, проверка на вырожденность треугольника, проверка на диапазон, 1 1 1 — равносторонний, 3 3 2, 3 3 10 и 3 случая на разносторонний.

При проведении тестирования крайне важно знать правильный результат, даже если этот результат получить сложно.

Если существует несколько независимых версий той или иной программы, то они должны добавить одинаковые результаты на одних и тех же тестах. Каждое выражение в программе при выполнении набора тестов должно быть выполнено хотя бы один раз.

Под нагрузочным тестированием понимается проверка устойчивости и производительности программного обеспечения в условиях предела его нормального функционирования. Необходимо производить тестирование на таблицах реального размера с реальным количеством пользователей.

Под стрессовым тестированием понимается проверка работоспособности программного обеспечения в условиях сильно хуже, чем условия нормального функционирования программы. Также к стрессовому тестированию относят проверку работоспособности в условиях ввода некорректных данных.

Стрессовое тестирование производится с обязательным учётом критичности бизнес-процесса. В общем случае при проведение стрессового тестирования от системы не ожидается, что она будет корректно работать при любых условиях, но ожидается, что она будет корректно информировать пользователя о своём состоянии и о ситуациях, связанных с некорректным вводом.

Часть проблем, связанных со стрессовым тестированием обычно проверяется при систематическом тестировании.

В настоящее время принято при большом количестве тестов использовать программы, которые последовательно последовательно запускают эти тесты. Часто эти программы пишутся на специализированных языках, разработанных специально для создания систем тестирования.

При использовании автоматизированных тестов активно применяются методики возвратного (регрессного) и замкнутого тестирования.

Возвратное тестирование выполняется всякий раз при внесении малейших изменений в программный код. Его цель — убедиться в том, что при внесении изменений в программу её поведение изменилось только в предусмотренных рамках.

Замкнутые тесты содержат в себе и вводимые данные, и ожидаемые результаты. Эти тесты только выдают информацию о том, успешно или не успешно они прошли.

Существуют также инструменты, которые взаимодействуют с программой во время её выполнения. Например, можно подсчитывать, сколько раз вызывалась та или иная функция, сколько раз выполнялся тот или иной оператор, во всех ли направлениях выполнялся каждый условный переход и т.д.

06.12.23

Тестирование, проводимое создателем кода или кем-то другим, имеющим доступ к исходному коду, называется тестированием белого ящика.

Обычно кроме разработчика код смотрят руководители и аналитики. Эти специалисты могут. Не только оценить качество кода, но и запустить некоторые тесты.

Тестирование, при котором тестер не имеет доступа к коду и не представляет себе его внутреннее устройство, называется тестированием чёрного ящика. Даже не видя код при тестировании тестировщик оценивает поведение программы на тех или иных тестах и может сделать некоторые наблюдения, которые сократят время отладки в будущем.

Даже после того, как программный продукт прошёл все этапы тестирования, он не вполне готов к окончательному внедрению. Поэтому его надо подвергнуть тестированию заказчиком (пользователем). Программы, которые передаются на тестирование пользователям, но не считаются полностью завершёнными, называются бета-версиями. От такого тестирования ожидается обнаружение каких-то новых ошибок, потому что пользователи могут опробовать программу неожиданными способами.

При этом недовольство пользователями интерфейсом, скоростью работы и т.д. необходимо купировать заранее другими способами.

12. Верификация и валидация ПО

Верификация и валидация являются видами деятельности, направленными на контроль качества ПО и обнаружение ошибок в нём. Имея общую цель, они отличаются источниками проверяемых в их ходе свойств, правил и ограничений, нарушение которых считается ошибкой.

Верификацией называется проверка соответствия, результатов, отдельных этапов разработки программной системы требованиям и ограничениям, сформулированным для них на предыдущих этапах. Кроме того, проверяется, что требования, проектное решение, документация и код оформлены в соответствии с нормами и стандартами, принятым в данном государстве, отрасли и организации при разработке ПО.

Обнаруживаемые при верификации ошибки и дефекты являются расхождениями или противоречиями между несколькими документами. При этом принятие решения о том, какой именно документ подлежит исправлению (может, и оба) является отдельной задачей.

Валидация проверяет соответствие любых создаваемых или используемых в ходе разработки и сопровождения ПО документов реальным нуждам и потребностям пользователей и заказчиков этого ПО с учётом законов предметной области и ограничений контекста использования ПО.

Эти нужды и потребности часто не зафиксированы документально, поэтому валидация является менее формализованной деятельностью, чем верификация. Валидация всегда проводится с участием представителей заказчиков, пользователей, бизнес-аналитиков или экспертов в предметной области — тех, чьё мнение можно считать достаточно хорошим выражением реальных нужд и потребностей пользователей, заказчиков и других заинтересованных лиц. Методы её выполнения часто используют специфические техники выявления знаний и действительных потребностей участников.

13.12.23

13. Характеристики качества ПО

С пользовательской точки зрения стандарт ISO предлагает следующую систему факторов качества программного обеспечения:

1. Эффективность (effectiveness) — способность решать задачи пользователей с необходимой точностью при использовании в заданном контексте
2. Продуктивность (productivity) — способность предоставлять определённые результаты в рамках ожидаемых затрат ресурсов
3. Безопасность (safety) — способность обеспечивать необходимо низкий риск нанесения ущерба жизни и здоровью людей, бизнесу, собственности или окружающей среде
4. Удовлетворённость пользователей (satisfaction) — способность приносить удовлетворение пользователям при использовании в заданном контексте

Основные факторы и атрибуты качества ПО с точки зрения его разработчиков:

1. Функциональность (functionality) — способность ПО в определённых условиях решать задачи, нужные пользователям. Атрибутами функциональности являются: пригодность, точность, совместимость, соответствие стандартам и правилам, защищённость
2. Надёжность (reliability) — способность ПО поддерживать определённую работоспособность в заданных условиях (защита от дурака). Атрибутами надёжности являются: среднее время работы без сбоев, устойчивость к отказам, способность к восстановлению, соответствие стандартам надёжности
3. Удобство использования или практичность (usability) — способность ПО быть удобным в обучении