Мещеряков, Давыдова – Языки программирования; Кауфман – Языки программирования, концепции и принципы; Орлов – Теория и практика языков программирования; Бэн-Али – Языки программирования, практический сравнительный анализ; Давыдов – Программирование и основы алгоритмизации; Гради Буч – Объектно-ориентированный анализ и проектирование; Кручинин – Технологии программирования; Ходошинский – Язык ПРОЛОГ в примерах и задачах.

Программа. Язык программирования. Стандарты языков программирования.

Язык программирования – набор правил, описывающих, какие последовательности символов составляют программу, какое вычисление производит программа; механизм абстрагирования, который даёт возможность описывать вычисления абстрактно, но одновременно это описание может быть приведено в понятную для процессора форму. Программа – последовательность символов, определяющих вычисление; запись алгоритма на языке программирования. Различные языки программирования выводят программиста на разные уровни абстракции: ассемблер – абстракция на уровне команд процессора; C, Pascal – уровень вычислений предметной области; объектно-ориентированные ЯП – абстракция на уровне объектов предметной области; SQL – абстракция баз данных; и т.д. Основной принцип абстрагирования: чем выше уровень абстракции, тем больше деталей скрыто от программиста.

Стандартизация ЯП. Стандарты делятся на частные и согласительные. Частный стандарт – определения, сделанные той компанией, которая разработала язык и имеет на него авторские права. Согласительный стандарт – созданные специальными организациями документы, основанные на соглашении всех заинтересованных участников. Этот стандарт является основным способом обеспечения единообразия различных реализаций языка.

Основные организации, занимающиеся созданием стандартов: ANSI, ISO, EEE.

Классификация ЯП. APL -> BPL -> C. ЯП низкого уровня. Особенности: оптимизированы под аппаратную структуру конкретного вычислительного устройства; программа представляла собой линейную последовательность элементарных операций с регистрами процессора. Низкоуровневое программирование – программирование, основанное на прямом использовании возможностей конкретной вычислительной системы. Для эффективного программирования на низких языках нужно знать структуру и функционирование системы в целом, организацию оперативной памяти, состав внешних устройств, адреса и форматы регистров, организацию и функционирование процессоров, способы адресации, систему команд, систему прерывания. Этапы развития низкоуровневых языков: машинный код, мнемокод, ассемблер, макроассемблер (расширение ассемблера за счёт включения макросредств, то есть средств определения и использования более мощных команд).

Языки высокого уровня. Отличия от низкоуровневых языков: абстрагирование от конкретных деталей аппаратного обеспечения, появление подпрограмм (повторное использование ранее написанных программных блоков), появление трансляторов (переводчик с одного ЯП на другой, в том числе и в машинный код), операторы и ключевые слова имеют более осмысленный вид для человека, высокая переносимость программ.

Все ЯП делятся на императивные и декларативные. Декларативные – не алгоритмические; императивные – процедурные, алгоритмические (C, Паскаль, Фортран). Декларативные подразделяются на логические (ПРОЛОГ) и функциональные (Lisp, Kaskel(?)).

ASM -> алгоритмический язык -> функциональные -> ООП -> логические -> естественные – по степени понятности человеком.

Процедурные языки – программа представляется как набор корректив, обращённых к процессору. Декларативные языки – программа представляет собой множество отношений между некоторыми сущностями решаемых задач. Декларативное программирование проще реализуется математическими средствами. Другое преимущество: высокая степень абстракции.

Функциональный подход (функциональные ЯП: Lisp, Haskell, Valid, VAL, LUCID, PLANIVER, ML). Отличительные особенности: программа может интерпретироваться как функция с одним или несколькими аргументами; сложные программы строятся посредством агрегирования (объединения) функций; типы отдельных функций могут быть переменными (различными); автоматическое динамическое распределение памяти (программист при необходимости запускает «сборщик мусора», который удаляет неиспользуемые данные); в таких ЯП легко реализуются рекурсивные функции. Недостатки: нелинейная структура программы; относительно невысокая эффективность реализации.

Логические языки (PROLOG, Mercury). Программа представляет собой совокупность правил или логических высказываний, также допустимы логические причинно-следственные связи (на основе импликации в основном). Базируются языки на классической логике и применимы для систем логического вывода. Преимущества: высокий уровень машинной независимости; возможность откатов (возвращение к предыдущей подцели при отрицательном результате анализа одного из вариантов) => увеличение эффективности реализации. Недостатки: ограниченное количество задач, к которым языки применимы; сложность эффективной реализации для принятия решений в реальном времени; нелинейная структура программы. Основное назначение таких языков – экспертные системы.

Объектно-ориентированное программирование (SMALLTALK, C, C++, C#...). Программа представляет собой описание объектов, их атрибутов,

их методов, совокупности объектов (класс), отношений между объектами. Преимущества: концептуальная близость модели к предметной области; поддержка механизма обработки событий, которые изменяют атрибуты объектов и моделируют их взаимодействие в предметной области; возможность повторного использования ранее написанного кода; полиморфизм объектов, классов и методов, что делает программу более гибкой и универсальной (возможность создания дочерних классов). Недостатки: сложность адекватной формализации объектной теории (возникают трудности тестирования и верификации созданного ПО).

Языки сценариев. Программа представляет собой совокупность возможных сценариев обработки данных, выбор которых инициируется наступлением того или иного события. События могут инициироваться как самой ОС, так и пользователем. Достоинства и недостатки: те же, что и у ООП; совместимость с передовыми инструментальными средствами автоматизированного проектирования; быстрая реализация ПО (CASE и RAD технологии). Примеры: VBScript; Power Script, Lotus Script, Java Script.

Языки поддержки параллельных вычислений. Программа представляет собой совокупность описаний процессов, которые могут выполняться как одновременно, так и в псевдопараллельном режиме (разделение времени между задачами). Преимущества: позволяют достичь заметного выигрыша при обработке больших массивов информации; возможно применение в системах реального времени. Недостатки: высокая стоимость реализаций ПО. Примеры: Ada, Modula, OZ, ML, SML, PL…

Классификации языков по типам задач. Задачи искусственного интеллекта: Lisp, Prolog, Haskell, Рефал, QAY, Planner… Задачи параллельного вычисления: Ada, Modula, OZ, ML, SML, PL… Задачи вычислительной математики и физики: Fortran, Kobol, Occam. Разработка интерфейсов: C, C++, Assembler, Java… – все, где есть работа с графическим интерфейсом. Оформление документов: HTML. Работа с БД: SQL.

Основные элементы языка программирования. Синтаксис – набор правил, которые определяют, какие последовательности символов являются допустимыми в языке. Синтаксис задаётся с помощью формальной нотации, например, расширенной формы Бекуса-Наура, которая используется для описания контекстно-свободных формальных грамматик. Все символы нотации делятся на терминальные и нетерминальные. Терминальные символы – минимальные элементы грамматики, не имеющие собственной грамматической структуры: предопределённые идентификаторы либо их цепочки. Нетерминальные символы – элементы грамматики, имеющие собственные имена и собственную грамматическую структуру. Нетерминальное выражение: идентификатор:: – выражение. , где выражение есть комбинация символов. Лексемма :: = описание. В РФБН используются описания: прямой жирный текст – фактические символы; курсив – синтаксические категории, помещённый в квадратные скобки элемент – элемент входит или не входит куда-либо (B::=[A] – B либо пустое, либо содержит A). Кампотинация A::=BC – один элемент состоит из двух. Выбор A::=B\C – A содержит либо B, либо C. Повторение A :: = {B} – A либо пустое, либо состоит из любой последовательности символов B. A :: = (BIC) (DIE) – группировка более простых выражений. A :: = [B] – условное вхождение (либо B, либо ничего). ‘…’, “…” – терминальная строка (текстовый элемент). (* … *) – комментарий. ? … ? – специальная последовательность символов. Например, множество чисел: list_numbers ::= () list_number ::= number.list_numbers.

program::= ‘PROGRAM’, white_space (разделитель), identifier, white_space, {assignment, “;”, white_space}; ‘END’;

identifier ::= alph_char, {alph_char|digit}; number ::= [“-“], digit, {digit}; string ::= ‘””, {all_char - ‘””}, ‘””; assignment ::= identificator, “:=”, (number|identificator|string); alph_chars ::= “A”|”B”…|”Z”; digit ::= “0”|”1”…|”9”; white_space ::= “ “; “\n”…; all_chars ::= ?all visible char?;

```
PROGRAM MY

x0 := 1;

y1 := “строка”;

VARIABLE := A_VARIABLE;

END
```

Основные ошибки, связанные с синтаксисом: ограничение на длину идентификатора; чувствительность к регистру; использование одинаковых символов для обозначения различных действий (присвоение и проверка на равенство).

Семантика – смысл синтаксических конструкций языка программирования. Семантика связана с особенностями самого языка и с концепцией программирования. Формальная запись семантики: s – состояние памяти в начале; S – команда; s’ – состояние памяти в момент t; s’ = p(S, s), где p – оператор воздействия на состояние памяти. Формальное описание семантики оператора S заключается в чёткой формулировке перехода из различных исходных s в соответствующий s’. Семантика более сложных конструкций может быть описана с помощью логических формул. if (c) s1; else S2; s’ = (c(s) => p(S1, s)) & ((-c(s)) => = p(s2, s)) c(s) – вычисление логического значения. Формализованный подход к семантике языка даёт возможность автоматизированного анализа программы.

Основные подходы к семантике: ориентированный на компиляцию (семантика – множество преобразований над синтаксической моделью) и ориентированный на интерпретацию (семантика – множество описанных на языке преобразований синтаксически правильных языковых конструкций). Общие требования к описанию языков программирования: полнота (синтаксис и семантика должны быть описаны для всех допустимых конструкций), ясность (все конструкции должны быть удобочитаемыми),

естественность (интуитивная близость языка к терминологии разработчика), реализм (язык должен учитывать ограничения на объём ОП, время реакции и т.д.).

Данные – сущности, над которыми выполняются вычисления и которые получаются в результате вычислений. Тип данных – множество значений и операций над этими значениями. Языки программирования предоставляют встроенные типы данных и возможность определения собственных типов данных. Все языки делятся на типизированные и не типизированные. Не типизированные языки – языки, в которых в любые переменные можно записывать любые значения в любом месте программы. Литерал – конкретное значение, заданное в программе буквально. Переменная – группа ячеек памяти, имеющая собственное имя и предназначенная для сохранения значений. Константа – переменная, которая не меняет своё значение после инициализации.

Операторы – команды языка программирования. Различают простые (присваивание) и составные операторы (условие). Основные операторы: присваивания, ветвления и перехода. Оператор присваивания: LEFT := RIGHT; - вычисление значения RIGHT, вычисление значения LEFT, которая трактуется как адрес ячейки памяти, запись значения RIGHT по адресу LEFT. Язык программирования обычно не выделяет порядок вычисления левой и правой части. На транслятор возлагается контроль за соответствие типов. Варианты действия транслятора: ничего не делать, записать как есть; может привести к получению пустых ячеек памяти или даже изменению ячеек, не привязанных к переменной; неявно преобразовать значение к типу переменной; строгий контроль соответствия типов (ничего не выполнять, выдать ошибку). Языки, где осуществляется полный контроль за типами, называются строго типизированными.

Управляющие операторы – операторы, отвечающие за последовательность выполнения операторов программы. К ним относятся ветвления, безусловные переходы, циклы, операторы выбора (case, switch).

Модули – крупные блоки программного кода. У модуля есть свой функционал. Модуль может представлять собой файл с исходным кодом или откомпилированный файл с определённым интерфейсом.

Среда программирования – набор инструментов, используемых для преобразования символов в выполнимые вычисления. Среда программирования состоит из редактора, транслятора (переводчик синтаксических конструкций исходного кода в объектный модуль), компоновщик или редактор связей (собирает объектные файлы отдельных компонентов программы и разрешает внешние ссылки от одного компонента к другому, формируя исполняемый модуль), загрузчик (копирует исполняемый файл с диска в память и инициализирует выполнение программы), библиотекарь (позволяет работать с библиотеками), отладчик (поиск ошибок: трассировка, контрольные точки и проверка измененияx данных), профилировщик (измерение трудоёмкости), средства тестирования, средства конфигурирования и версионности (простановка новых версий программы), средства автоматизированной генерации кода и средства визуальной разработки.

трансляторы

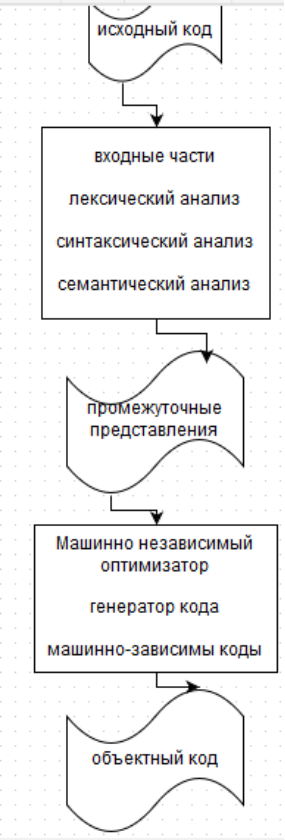
- компиляторы – переводит все исходный код в объектный за раз. C,…
  - достоинства – многие ошибки выявляются еще до запуска программы

- выполнение программы происходит быстрее
- высокая защита исходного кода
- переносимость на уровне процесса

- интерпретаторы – выполняет перевод по командно в объектный код непосредственно перед выполнением. Интерпретатор абстрактной машины занимается этим. Java
  - достоинства – не требуется дополнительных преобразований, более простая реализации программы
  - запуск программы происходит быстрее
  - не требует наличия всех компонентов программы
  - переносимость на уровне абстрактной машины

Процесс трансляции

Компилятор



Интерпретатор

Лексический анализатор – преобразует последовательности символов в синтаксические инструкции. Синтаксический анализатор – строит синтаксические иерархические структуры (связывает отдельные компоненты).

Семантический анализатор – предает смысл синтаксическим инструкция и иерархиям (определяет адреса переменных, подпрограмм, последовательность параметров и так далее).

В итоге в обоих случаях получается некое промежуточное представление (абстрактный код) который с точностью до идентификаторов однозначно соответствует исходному коду.

Оптимизатор – пытается убрать лишние операции, заменить медленные инструкции на более быстрые и так далее. Отладку лучше проводить при выключенной оптимизации, а тестирование и при выключенной и включенной.

Оптимизация промежуточного представления:

(1+x)(y-7)+(5+x)(y-7)

Машинно-ориентированная оптимизация:

использование регистров и кэш памяти вместо оперативной.

Локальная оптимизация:

замена нескольких команд более эффективными командами.

Этапы семантического анализа

1. проверка соблюдения во входной программе семантических соглашений (заключается в сопоставлении входных цепочек исходной программы с требованиями семантики входного языка);
2. дополнение внутреннего представления программы (связано с дополнением в него операторов и действий неявно предусмотренных семантик входного языка);
3. проверка смысловых норм языков программирования (обеспечивает проверку компилятором соглашений, выполнение которых связано со смыслом как всей программы в целом так отдельных ее частей);

Типизация и типы данных

Все данные хранятся в ячейке памяти. Объем одной ячейки кратен одному байту.

Машинное слово – минимальный для процессора объем одной ячейки.

Элемент данных, рассматриваемый как единое целое в некий момент выполнения программы, называют объектом данных.

Атрибуты объектов данных:

1. тип данных – значение объекта, операции применяемые к нему
2. место положения – хранит координаты области памяти, отведенной под объект данных
3. текущая величина объектов данных
4. имя
5. принадлежность – сведения о принадлежности данного объекта к другим объектам данных
6. время жизни
7. область видимости – фрагмент программы в пределах которого доступен объект

Сумму правил связанных с типом данных и их экземплярами объединяют в понятие система типизации данных.

Простые типы данных:

Целочисленные типы данных

int, integer…

signed, unsigned

числа со знаком представляются дополнительным кодом

максимальное число которое можно записать в переменную: $2^{w-1} - 1$, где $w -$ разрадность слова

byte – 1 байт = char

0..255

signed int(shortint)

int

longint

size_t – для хранения индексов массива, гарантирует не переполнение

Целочисленные операции выполняются быстро в большинстве случаев независимо от размера.

неэффективно использовать переменные размера больше чем размер регистра, обычно int - наиболее эффективен, если нет опасности переполнения.

Знаковые и безнаковые целые

разница есть в случае деления, преобразованиях к вещественному значению, различное поведение при переполнении.

Преобразования между знаковыми и безнаковыми не стоят ресурсов процесса

нельзя сравнивать знаковые и безнаковые значения.

простые операции занимают один такт в большинстве микропроцессоров

операции инкремента и декремента занимают один такт процессора

09.10.2015

Логический тип Boolean

0 или 1, false or true

Порядок логических операндов с лева на права по принципу ленивых вычислений.

При смене мест операндов в логических выражениях следует опасаться побочных эффектов.

Логические переменные переопределены.

Вещественные числа

Два подхода к двоичному представлению

1. согласно двоичной арифметике

a. двоично-кодированные десятичные числа (BCD)

2. кодировать цифры числа как целое число, дополняя информации о позиции десятичного разделителя

a. числа с фиксированной точкой

i. плюсы:

ii. количества знаков после запятой определяет абсолютную погрешность;

iii. недостатки

iv. относительная погрешность переменная

v. в такой формат нельзя вписать некоторые числа

b. числа с плавающей точкой $X10^m$ $X[0,1)$ $m-$ порядок 0,125 E-5

i. плюсы

ii. количество знаков мантиссы определяет относительную погрешность

iii. очень широкий диапазон чисел + низкая относительная погрешность

Основные ошибки

Переполнение – порядок больше верхней границы диапазона.

Потеря значимости – порядок ниже нижней границы.

Исчезновение операнда – операнд может исчезнуть, если он мал относительно другого операнда.

Умножение ошибки – многократное увеличение абсолютной погрешности операнда.

Символьный тип данных (char,byte)

ASCII, VTF8

Перечисление

Требуется явная типизация значения типа перечислений.

Связывание переменных

Связывание – процесс установления связи между атрибутами.

Связывание типа- имя переменной связывается с типом данных

Связывание значений – имя связывается со значением.

Статическое связывание (раннее) выполняется в период компиляции

Динамическое связывание происходит в период выполнения программы.

Статическое связывание

Делается за счет явных и неявных объявлений

Явное объявление – оператор программы, сообщающий компилятору сведения об именах и типах объектов данных. Место размещения в программе определяет время жизни.

Неявное объявление – объявления по умолчанию. Применяются при отсутствии явных объявлений.

Динамическое связывание

Происходит в период выполнения программы

Оператор объявления переменной не содержит имени типа, тип определяется при присвоении ей оператором присваивания.

Любой переменно может присвоено быть любое значение в любой момент времени.

Недостатки – снижается надежность программы (возможность определения ошибок становится меньше)

высокая стоимость программы в плане ресурсов (высокая стоимость реализации особенно во время вычислений)

Применяется в интерпретированных языках

Типы переменных в зависимости от связывания

1 Статической называют переменную, которая связывается с ячейкой до начала выполнения программы и сохраняет связь до окончания выполнения.

2 Стековые – связывание с памятью осуществляется при обработке операторов объявления переменных, а типы переменных связываются статически

3 Явные динамические – безымянные ячейки памяти, размещаемые и удаляемые с помощью явных операторов программ. Обращаться к ним можно при помощи указателей и ссылок, сами переменные хранятся в динамической памяти.

4 Неявные динамические переменные – связываются с динамической памятью только при присваивании им значения.

Контроль типов

1 Динамический – во время выполнения.

2 Статический – во время компиляции.

Алгоритм:

1 компилятор собирает всю исходную информацию в таблицу символов;

2 проверка всех операции программы на предмет правильности типов операндов;

3 после проверки операндов i-ой операции определяется тип ее результата и сохраняется для проверки следующей операции;

Динамический контроль типов

Плюсы

1 гибкость программы;

2 можно создавать настраиваемые программы способные работать с данными любого типа.

Минусы

1 понижение надежности вычислений;

2 Снижение скорости вычислений;

3 возрастание накладных затрат на память.

Составные типы данных

Массивы

лекция за 16.10.2015

Массив – структура данных, содержащая последовательность элементов одинакового типа. Атрибуты массива: количество элементов, тип данных, список значений индексов.

Различают компоновку массива и размещение массива в памяти.

При компоновке задаются относительные адреса элементов массива и вычисляются формулы для адресации элементов (имеется адрес первого элемента, и относительно него размещаются остальные).

При размещении определяются действительные, машинные адреса элементов и выделяется память под эти элементы. A[i] = (base-low*w)+i*w, base – адрес начала массива, w – длина элемента, low – адрес первого элемента. Если адрес первого элемента равен нулю, то получается случай A[i] = base+i*w. Время доступа к элементам массива не зависит от номера элемента. Значение индекса должно находиться в пределах допустимого диапазона. Разновидности массивов по типу связывания: связывание типа индекса массива с переменной массива обычно выполняется статически, но по диапазону индексов связывание происходит зачастую динамически. Всего выделяются пять разновидностей массивов: статические (массив со статическим связыванием по диапазонам индексов и таким размещением в памяти, которое происходит до начала работы программы; эффективны по причине отсутствия затрат времени на динамическое размещение в памяти; память ими занимается на всё время выполнения программы), явные стековые (массив со статическим связыванием по диапазонам индексов и таким размещением в памяти, которое происходит по итогам обработки объявления в ходе выполнения программы; повышается эффективность использования памяти; дополнительные затраты времени на размещение и удаление из памяти), стековые (динамическое связывание по диапазонам индексов и динамическое размещение в памяти, которое происходит в ходе обработки объявления; связанность сохраняется в течение всей жизни переменной; не требуется знать заранее размер массива; затраты на размещение массива в памяти), явные динамические (связывание и по диапазонам индексов, и по памяти происходит после размещения в памяти; память выделяется в куче (динамическая память), а не в стеке; легко изменяется размер массива; на размещение в куче тратится больше времени, чем в стеке), динамические (массив с таким связыванием по диапазонам индексов и размещением в памяти, которое повторяется многократно в течение всего цикла жизни массива; максимальная гибкость; многократное размещение и удаление из памяти). В C++: static – статические, new и delete

– работа с динамическими массивами, malloc и free – работа с явными динамическими массивами, остальные – явные стековые. Прямоугольный массив – многомерный массив, в котором строки и столбцы имеют одинаковое количество элементов. Массив массивов (невыровненные массивы) – многомерный массив, в котором не требуется одинаковой длины. Ассоциативный массив – неупорядоченное множество элементов данных, индексированных таким же количеством величин, которые называются ключами (Perl, Python, Ruby, Loa).

Строки символов. Строка – последовательность символов (либо отдельный тип, либо массив символов). Разновидности строк: строка фиксированной длины (если строка не влезает, то она усекается, если строка слишком короткая, то пустые поля забиваются каким-либо символов), строки переменной длины не более определённого максимума (концом строки считается нулевой символ, текущая длина строки переменна, но не входящие в строку символы отсекаются). Основные операции: конкатенация; сравнение (отношения над строками), выделение подстроки по индексам, выделение подстроки на основе сопоставления с образцом (определяющая структура данных называется регулярным выражением).

Записи, структуры – неоднородные по типу структуры данных, в которых отдельные элементы идентифицируются именами. Атрибуты: количество полей, тип данных для каждого поля, имя. Допускается создание структуры из массивов или массива структур, и даже структуры структур.

Множество – структура, содержащая неупорядоченный набор различных значений. Все элементы множества должны быть одного и того же простого типа.

Кортеж – структура данных из разнотипных элементов, не имеющих имён и представляемых значениями. Кортежи могут конкатенироваться, могут удаляться.

Списки – упорядоченная последовательность некоторых структур данных. Первый элемент списка называется головой списка, остальные

являются хвостом. Списки бывают однородные и неоднородные, как правило, типы элементов списка не объявляются явно. Списки редко имеют фиксированную длину.

23.10.2015

Организация повторения операторов

Две схемы – итерация и рекурсия

Итерация – способ многократного повторения операторов, реализуемых циклом.

Рекурсия – такой способ организации повторений операторов при которых последовательность операторов вызывает сама себя непосредственно либо с помощью других последовательностей.

Циклы

С постусловием и предусловием

Цикл со счетчиком

Есть переменная цикла, которая принимает дискретное количество значений и используется для подсчета количества повторений тела.

Цикл повторения по данным

позволяет перебирать все элементы многоэлементной структуры обращаясь к каждому из них по очереди.

Циклы в функциональных языках

Вместо итераций используют рекурсию

F#

```
cet rec loopfor bodyloop reps=
    if reps <=0 then
        ()
    else
    bodyloop ()
loopfor bodyloop, (reps+);
```

Моделирование цикла со счетчиком:

Счетчик может быть параметром для функции, многократно выполняющей тело цикла. Тело определяется во вспомогательной функции, которая тоже посылается в функцию цикла как параметр.

Бесконечные циклы

C

for (; ;) {…}

continue – прерывает текущую операцию и начинает следующую.

Особенности для C

Внутри всех выражений могут быть присваивания

Все выражения не обязательны

Не существует явных счетчиков и параметров цикла

Все переменные, указанные в секи управления могут изменяться в теле цикла

В качестве первого и третьего выражений разрешают применять составные выражения

C++

Для управления циклом может использоваться как и арифметическое выражение так и булево.

Первое выражение может содержать определение переменных.

Рекурсии

Два важных элемента

Должно быть условие прекращения рекурсии

Вызов с другим значением параметра (чтобы не произошло зацикливание нужно чтобы внутренний вызов производился с модифицированным значением параметра).

Виды рекурсий

1. Линейная рекурсия – выполнение тела подпрограммы приводит не более чем одному рекурсивному вызову за один рекурсивный срез

2. Повторная рекурсия – частный случай линейной рекурсии, когда рекурсивный вызов внутри подпрограммы является последним оператором, отсутствуют предварительные и отложные вычисления

3. Каскадная рекурсия – Возникает, если рекурсивные вызовы могут возникнуть более одного раза

4. Взаимная рекурсия – Циклическая последовательность взаимных вызовов нескольких подпрограмм (вторая к первой первая ко второй)

5. Удаленная рекурсия – В теле функции и рекурсивных вызовов в выражениях являющихся фактическими параметрами снова встречаются рекурсивные вызовы этой функции. f(f(2))

Подпрограммы

Группа операторов, которая может быть выполнена неоднократно

То есть можно рассматривать как:

1. Сегмент программы, который должен выполняться на разных стадиях вычислений

2. Логическая единица декомпозиции программы.

Подпрограммы

1) Функции

2) Процедуры

Основные понятия

Интерфейс подпрограммы – указание ее имени и параметров и типа возвращаемого значения.

Имя – любой допустимый идентификатор языка (нужен для однозначного восстановления соответствия между вызывающим и вызываемым кодом)

Формальный параметр – некоторое значение, передаваемое внутрь подпрограммы и использующееся для ее вычислений

Тело – группа операторов из которых состоит подпрограмма.

Вызов подпрограммы – ОПЕРАТОР, указывающий на необходимость ее выполнения.

Подпрограммы могут быть вложенными (нужно учитывать область видимости).

Параметры

1) Формальные – разновидность локального объекта данных в подпрограмме, находятся в объявлении подпрограммы.

2) Фактические – могут выступать локальные переменные, арифметические выражения, формальные параметры, константы, глобальные переменные, массивы, структуры, результаты вычислений функций.

Преимущества подпрограмм

1. Уменьшение сложности программирования;
2. Закрытость реализации (модификация алгоритма внутри подпрограммы не воздействует на остальную часть программы);
3. Модульность программ
4. Расширение возможностей языков программирования

Методы передачи параметров

30.10.2015

Передача параметров – сопоставление между фактическими и формальными параметрами.

Методы передачи параметров – способы, которыми параметры передаются в подпрограмму или возвращаются из нее.

Два способа для связывания параметров

Позиционное сопоставление (позиция) – устанавливается на основе позиций параметров в списках фактических и формальных параметров.

Параметр по умолчанию

Пример python

```
def salary(hour, tax_free=, hour_rate)
    salary (120, hour_rate=10.0)
```

Сопоставление по имени – при вызове подпрограммы можно явно указать какой формальный параметр должен соответствовать данному фактическому, обращаясь по имени

Пример Python

```
adder (size= the_size,
    list=the_list,
    sum=the_sum)
```

В С только позиционное

Ruby

```
list=[1.2.3.4]

def funcR (p1.p2.p3.p4)

…

end

…

funcR (‘teacher’, growth=>175,weight=>70,age=>30,*list)
```

В результате

```
p1=teacher

p2={growth=>175,weight=>70,ageY}

p3=1

p4=[2.3.4]
```

Методы передачи параметров:

1) Передача параметров по значению

P(x)

x – форм

P(E)

E – фактич.

формальный параметр x подпрограммы P(x) получает значение фактического параметра E, при этом при вызове P(E) выполняются следующие действия: для параметра x выделяется место во временной памяти; вычисление E; x=E; разрывается связь между формальным и фактическим параметрами; выполнение тела подпрограммы; возвращение результата.

Преимущества

фактический параметр может быть выражением

с помощью формального параметра нельзя вернуть результат в точку вызова (больше отрицательная хар-ка)

любые изменения значения формального параметра теряются после выполнения подпрограммы

В языках Pascal, C и других этот метод является основным

2) Передача параметров по ссылке

Формальный параметр превращается в синоним места размещения фактического параметра (подпрограмме становится доступен указатель на местоположение этого объекта).

Алгоритм

Этапы:

1) В вызывающей программе определяются указатели на объекты данных соответствующие фактическим параметрам (список указателей сохраняется в общей области памяти, доступной также и вызываемой подпрограмме), управление передаются вызываемой подпрограмме.

2) В вызванной подпрограмме список указателей на фактические параметры используется для вычислений с целью получения значения этих параметров

Следствия

1) фактический параметр значения может быть выражением, фактический параметр-ссылка выражением быть не может

2) Место размещения фактического параметра ссылки вычисляется и передается до выполнения тела процедуры

C++

```
vc:d swapc (int, *px, int *py) {
    int z;
    z=*px; *px=*py; *py=z;}
    swapc( &a,&b)
```

Достоинства

1) Высокая эффективность с точки зрения времени и памяти (не требуется копировать значения фактических параметров, а затем отдельно обновлять их

Недостатки

1) используется косвенная адресация (доступ к обрабатываемым параметрам замедляется)

2) Понижается безопасность, понижение безопасности хранения фактических параметров, поскольку двусторонний канал связи между ними открыт на все время работы подпрограммы

3) Возможность возникновения псевдонимов фактических параметров

3) Передача по значению результата – в начале все фактические параметры копируются в формальные параметры, а в конце обратно, при этом фактические параметры выражения передаются по значению, фактические параметры имеющие место размещения обрабатываются следующим способом: copy-in – вычисляются значения и места размещения

фактических параметров, значения присваиваются формальным параметрам, места размещения сохраняются до этапа copy-out; на втором этапе выполнения тела подпрограммы и обработка параметров; copy-out – конечные значения формальных параметров копируются обратно в места размещения сохраненных на этапе copy-in

Достоинства

1) Не используется косвенная адресация – обработка параметров ускоряется

2 Безопасность хранения фактических параметров повышается, так как во время выполнения тела подпрограммы канал закрыт

Недостатки

1) Необходимость хранить параметры в нескольких местах и тратить время на копирование их значений и адресов

2) Сохраняется проблема псевдонимов

4) Передача параметров по результату

параметр передаваемый по результату используется только для передачи результатов вычисления назад в вызывающую программу

Начально значение фактического параметра не имеет никакого значения для подпрограммы и ею не используется, формальный параметр – локальная переменная, у которого начальное значение отсутствует, когда подпрограмма завершается, конечное значение формального параметра присваивается фактическому параметру.

Недостатки

1) Порядок может задаваться реализаций языка

2) выбор момента для вычисления адреса фактического параметра (он может определятся в начальной фазе вызова подпрограммы или при возвращении и нее (определяется разработчиком реализации языка)

Реализация методов передачи параметров

В большинстве современных языков происходит через стек выполняемой программы.

Параметры, передаваемые по значению заносятся в ячейки стека, далее эти ячейки хранят соответствующие формальные параметры

По результату значения – фактические параметры помещаются в стек откуда извлекаются вызывающим программным модулем

По значению результата – комбинация передачи по значению и результату

По ссылке – независимо от типов фактического параметра, в стек помещается только его адрес.

Указатели и динамическая память

Распределение памяти

Код Константы	Статические данные	куча		стек

Код – машинные команды.

Константы – небольшие константы, содержащиеся внутри команд.

Стек – стековая память, используемая для записи и активации, которая содержит параметры переменные и ссылки.

Статические данные – переменные, объявленные в главной программе и других местах.

Куча – область данных, их которых данные динамически выделяют специфическими командами на языках.

Менеджер кучи – компонент исполняющей системы, который выделяет и освобождает память.

Если память не очищать, то возникает проблема фрагментации.

06.11.2015

Указатели

Указатель – переменная, значением которой является адрес другой переменной.

int i = 4

int *ptr = 8i

Указатель предоставляет косвенный доступ к элементам известного типа

Объект на который указывает указатель называется указуемым или обозначаемым

Операции над указателями

1) Разыменования в C * - получаем объект на который ссылается указатель

int *p

float *pf

2)& - получение адреса переменной (получение адреса указателя)

Разница между переменой указателем и указуемым объектом

int i1 = 17

int i2 = 20

int *ptr1 = &i1

int *ptr2 = &i2

*ptr1 = *ptr2 //обе переменные указатели будут иметь одинаковое значение.

Указатель константы и указатель на константу

int i1, i2

int *const p1 = &i1 //указатель константы – переназначить на другую область памяти нельзя, но значение переменной менять можно

cons tint *p2 = &i2 //нельзя изменить значение

Типизированный и не типизированный указатель

void *void_ptr

int * int_ptr

Типизированные указатели неявно могут быть преобразованы в указатели на void, но не обратно

char * char_ptr;

void_ptr = int_ptr; //true

char_ptr = void_ptr //C – ошибка C++ - warning

char_ptr = int_ptr //C – warning C++ - ошибка

Зачем нужны указатели

1) Повышается эффективность (вместо копирования или пересылки можно пересылать только указатель на эту структуру)

2) Динамические структуры данных

3) Доступ к большим структурам данных (используется для косвенного обращения к большим структурам данных)

Функции для работы с динамической памятью

malloc ()- Выделяет память

void free (void *p);

free() – освобождение

void *malloc (size_t кол-во байтов) //#include <stflib.h>

Пример

```
p = malloc (128)
if (!p) {printf(“error”);
                exit(1);
}
free (p);
```

Если free вызвать с ошибочным аргументом, то разрушается вся система динамического распределения памяти.

C++

new

delete

```
int *ptr2 = new int[251]
delete [] ptr2
```

Утечки памяти и повисшие указатели

Память, которая распределена, но недоступна, называется мусором. Если программа создает мусор, то она дает утечки памяти

Повисший указатель – есть указатель на память которой используется для других целей

Способы борьбы

1) Ручной – использование delete

2) Автоматический – сборщик мусора

Исключительные ситуации

Исключение – необычное аварийное событие которое обнаруживается аппаратно или программно и требует специальной обработки.

Обработчик исключений – специализированная подпрограмма, которая реагирует на аварию и в максимальной степени старается устранить последствия исключений.

Требования к аппарату исключений:

1) Полнота исключений на любое исключение должно быть предусмотрено

2) Минимальность возмущений –

3) Минимальность повреждений – ущерб при возникновении исключений должен быть минимальный

Характерные признаки участков кода в которых нужно использовать исключений обработчики

1) Возможность возникновения регистрируемых ошибок, включая ошибки системных вызовов. когда необходимо организовать устранение этих ошибок

2) Интенсивное использование указателей

3) Интенсивное использование данных в виде массивов

4) Арифметические операции с участием вещественных чисел

5) Наличие вызова функций, которые могут генерировать исключения либо программным путем, либо в силу того, что их работоспособность не была должным образов проверена.

Исключения

1) Синхронные – могут возникнуть в определенных заранее известных местах программы

2) Асинхронные – могут возникнуть в любой момент времени, не зависит от того какую инструкцию выполняет система

1) Предопределенные – заранее встроены в язык программирования

2) Определяемые – определить свой класс исключений, соответствующих нашей задаче.

Исключения естественным образом группируются в семейство

Этапы работы с определяемыми исключениями

1) Определение исключения, ловушки, генераторы исключений

2) Возникновение исключения – нормальные вычисления приостанавливаются и система переключается режим обработки исключений

3) Обработка исключений

3.1) Распространение исключения – распространяется от точки его порождения до точки его обработки, при этом используется принцип динамической ловушки (выбирается обработчик динамически ближайший к месту происшествия)

3.2) Реакция на исключение

4) Продолжение вычислений

Две схемы обработки

1) Обработка с возобновлением – обработчик исключения ликвидирует возникшую проблему и приводит программу в состояние, когда она может работать дальше по основному алгоритму (типично для обработки асинхронных исключений)

2) Обработка без возобновления – после выполнения кода обработчика управление передаётся в некоторое заданное заранее место программы

Многоуровневая система исключений

Иерархическая структура

Обработка исключений

try{(код)

throw (raise)} (выбрасывание)

catch (expect) //формальный параметр

В Питоне else(выполняется если исключение не обнаружено)

finally (всегда выполняется)

Особенности для C++

1) Нет предопределенных исключений, но есть стандартные библиотеки, которые определяют и генерируют исключения

2) Обрабатываются только исключительные ситуации, явно генерируемые некоторой функцией

3) поддерживается окончательная модель обработки – при возникновении исключения работу невозможно продолжить с места исключения

4) обработка исключений возможно только в функции, вызванной до возникновения ее и еще не завершившийся

5) Если обработчик не обнаружен, тогда вызывается стандартный обработчик unexpected

6) Если обработчик поймал исключение. то обработка этого исключения другими подомными обработчиками, которые могут для него существовать, невозможна

7) Если управление передано catch блоку, то исключение считается обработанным

8) Обработчик как и любая функция может заявить исключение

Схема Бертрана – Мейера

Реализация схемы обработки исключения с возвратом

Полиморфизм в языках программирования

- возможность объектов с одинаковой спецификацией иметь различную реализацию.

Различают полиморфные объекты и полиморфные функции

Полиморфный объект – сущность (переменная, аргумент функции и так далее) хранящая во время выполнения программы значение различных типов.

Полиморфные функции – функции которые имеют полиморфный аргумент.

Различают статический полиморфизм и динамический

В статическом полиморфизме множественные формы конкретизируется на этапе компиляции, а в динамическом структурная неопределенность остается до этапа выполнения

Статический полиморфизм

Преобразование типов

Операции преобразования типов – операции преобразования значения одного типа другому.

int i = 5

float f = i

Существует два варианта преобразования типов:

Перевод значения одного типа к **допустимому** значению другого типа

Пересылка значения как не интерпретируемой строки битов. Разрешает программе использовать одну и ту же строку битов разными способами.

Пример явного приведения C++

int n;

float f;

f = (float)n;

n = (int)f;

Пересылка значения как не интерпретируемой строки битов

f = *((float*)&n) //первой выполняется &, преобразовали тип к полученному указателю, разыменовали.

Явное приведение типов в C++

static_cast //обычное статическое приведение типов

var_int = static_cast<int>(var_float)

const_cast //преобразует тип операнда с пометкой const в аналогичный без пометки const

const Key *ptr_Key = *Key;

Key *myPtr = const_cast <Key*> (Ptr_Key)

reinterpret_cast осуществляет низкоуровневую пере интерпретацию битов его операнда (жестко машинно зависимое преобразование)

char *ptr_char = reinterpret_cast <char*> (str_int)

dynamic_cast // применяется для преобразования ссылки или указателя на объект базового класса в ссылку или указатель на объект другого родственного класса

Преимущества неявного приведения типов

1) Освобождение программиста от рутинной работы

Недостаток

1) Может ввести к скрытым ошибкам

Перегрузка – возможность использования одноимённых подпрограмм или операторов, различающихся типом или количеством параметров в пределах одной области видимости

Применяется в статически типизированных языках. которые проверяют типы аргументов при вызове функции.

Механизм реализации:

1) компилятор должен идентифицировать функцию не только по имени но и по параметрам

2) должны быть введены специальные синтаксические конструкции вида

<операнд 1> <знак операции> <>

3) Нужно разрешить описывать поведение операторов в виде функции

```
int add (int a; int b)
                {return(a+b)};
int add (float a; float b)
                {return(a+b)};
int add (int a; int b, int c)
                {return(a+b+c)};
add(1,2);
add(1,2,3);
```

\+ Удобства

Минусы add (0,5,1)

1) идентификация, если нет подходящего образца, решение – требовать однозначный подходящий вариант перегруженной функции

выбирается перегруженная функция не приводящая к потери информации, если происходит потеря информации тогда ошибка, если есть более одного варианта приведения типов, то выдать ошибку.

2) приоритет и ассоциативность – порядок выполнения и следования операндов

Если приоритет и ассоциативность заданы жестко, то это может не соответствовать предметной области.

Родовые настраиваемые сегменты и шаблоны

В C++

template < параметры шаблона >

объявление функции которая может иметь параметры шаблона

Пример

```
template < typename T >
T sqr (T x)
{return x*x;}
int a=y
```

int x = sqr(a)

20.11.15

Вариантные и неограниченные записи

Запись и структура эквивалентны

Вариантная запись – запись, которая состоит из фиксированного числа полей, но позволяющая по-разному рассматривать область памяти занимающей полями

```
typedef enum{ recor_Code,
                Array.Code} Codes;
typedef struct {
Codes code;
union {
Arr a;
Rec r;} Data;
}S_Type;
S-Type S;
S_type s;if ( s.code == Array_Code) i=s.data[4];
else s.data.r:h
```

Унифицированный язык моделирования UML

UML – язык графического описания для объектного моделирования в области разработки программного обеспечения. Является языком широкого профиля, это открытый стандарт, использующий графические обозначения для создания абстрактной модели системы.

Назначение UML

Создан для определения визуализации проектирования и документирования в основном программных систем

Моделирование бизнес процессов

Системное проектирование

Отображение информационных структур

Типы UML диаграмм

- Диаграмма активности (деятельности
- классов
- связей
- компонентов
- составных структур
- развертывания
- обзора взаимодействия
- объектов машинного состояния

Диаграмма активности – представление алгоритмов неких действий (активностей), выполняющихся в системе

Дорожки объектов – часть области диаграммы деятельности на которой отображаются только те деятельности, за которые отвечает конкретный объект.

Диаграмма прецедентов

Цели создания

определение границы и контекста моделируемой предметной области на ранних этапах проектирования

формирование общих требований к поведению проектируемой системы

разработка концептуальной модели системы для ее последующей детализации

подготовка документации для взаимодействия с заказчиками и пользователями системы

основные элементы

Эктор - множество логических связанных ролей исполняемы при взаимодействии с прецедентами или сущностями. Эктором может быть человек или другая система подсистема или класс, которые представляют нечто вне сущности

Прецедент – описание множества последовательных событий выполняемых системой, которые приводят к наблюдаемому эктором результату

Диаграмма состояний

Применяется для того чтобы объяснить каким образом работают сложные объекты и как объект переходит из одного состояния в другое

Объекты характеризуются поведением и состоянием, в котором находятся

Состояние – ситуация в жизненном цикле объекта, во время которой он удовлетворяет некоторому условия выполняет определенную деятельность или ожидает какого-то события. Состояние объекта определяется его атрибутами.

Диаграмма классов описывает статическую структуру системы, показывая ее классы, их атрибуты и методы ,и также взаимодействия этих классов

Объектно Ориентированное Программирование

Выделяют объектно ориентированное программирование, анализ и проектирование, на результатах ООА формируются модели, на которых основывается ООП. ООПр создает фундамент для конкретной реализации системы с использованием методологии ООП

Язык является объектно ориентированным если выполняются следующие условия

1) Поддерживаются объекты. То есть абстракция данных имеющей интерфейс в виде именованных операций и собственные данные с ограничением доступа к ним

2) Объекты относятся к соответствующим классам (типам)

3) Классы (типы) могут наследовать атрибуты суперклассов (супер типов)

Преимущества объектной модели

1) Позволяет использовать возможности ООП

2) Повышает уровень унификации разработки и пригодность для повторного использования тоже повышается

3) Приводит к построению систем на основе стабильных промежуточных описаний, что упрощает процесс внесения изменений

4) Уменьшает риск разработки сложных систем

5) Ориентирована на человеческое восприятие мира

Классы и объекты

Класс - абстрактный тип данных снабженный некоторой реализацией

Есть интерфейсы и реализация

Интерфейс – то что мы видим

Класс который полностью реализован называется эффективным

Класс который реализован лишь частично или не реализован вовсе называется отложным

реализация отвечает за внутреннее представление класса, а интерфейс за внешнее представление класса

Атрибуты определяют состав и структуру данных. хранимых в объектах класса

Методы – операции которые должны выполнять объекты данного класса

Три вида специальных классов:

1)Абстрактный класс – который не имеет экземпляров (объектов), в нем объявляются методы, но не реализованы, на основе этого класса делаются наследники, которые уточняют данную абстракцию

2)Интерфейс – абстрактный класс который содержит только объявление методов и статические константные поля

3) Утилита – класс в котором присутствуют только статические члены, используется для группировки наиболее часто используемых алгоритмов

27.11.2015

отношения между классами

1) Ассоциация показывает структурные отношения межд объектами экземплярами класса т е соединения между классами

когда класс участвует в ассоциации он играет в этом отношении определенную роль

один к одному

один ко многим

многие ко многим

2) Обобщение – отношение между общей сущностью и специализированной разновидностью этой сущности

Наследование – это отношение при котором один класс разделяет структуру и поведение определенные в одном другом (простое наследование) или во многих других (множественное наследование) классах

3) зависимость – это отношение которое показывает что изменение в одном классе (независимом) может влиять на другой класс (зависимый) который использует его

С помощью зависимости уточняют какая абстракция является клиентом а какая поставщиком

4)Реализация – это отношение между классами в котором класс-приемник выполняет реализацию операций интерфейса класса- источника

5)Агрегация обозначает отношения классов в иерархии целоя /часть. Говорят что агрегация образует part-of иерархию классов (и объектов)

Агрегация обеспечивает возможность перемещения от целого (агрегата) к его частям (атрибутам)

Нефизическое включение - агрегация

6) Композиция

Объект – это экземпляр некоторого класса

При реализации объекта для атрибутов будет выделена память необходимая для хранения всех атрибутов

Каждый атрибут будет иметь конкретное значение в любой момент времени работы программы

Объектов одного класса в программе может быть сколь угодно много, все они имеют одинаковой набор атрибутов описанный в классе но значения атрибутов описанный в классе но значения атрибутов у каждого объекта дальше в методе

Свойства объектов

1) индивидуальность – это характеристика объекта которая отличает его от всех других объектов

2) состояние – характеризуется перечне всех атрибутов и текущими значениями каждого из них

3) поведение – это его деятельность с точки зрения воздействия на другие объекты или подверженности воздействиям со стороны других объектов

Операции клиента

1) модификатор – изменяет состояние объекта

2) селектор – дает доступ к состоянию, но не изменяет его

3) итератор – доступ к содержанию объекта по частям в строго определенно порядке

4) конструктор – операция создания объекта и/или его инициализации

5) деструктор – операция освобождающая состояние объекта и/или разрушающая сам объект

отношения между объектами

1) контроллер – объект, который может воздействовать на другие объекты, но сам не подвержен воздействиям

2) сервер – объект, который никогда сам не воздействует на другие объекты

3) агент объект который может как воздействовать на другие объекты так и использоваться ими. Агент создаётся для выполнения работы от имени контроллера или другого агента.

Основные принципы ООП

Абстрагирование – это способ выделить набор существенных характеристик объекта исключая из рассмотренния незначимы которые позволяют отличить его от всех других видов объекта (определит его концептуальные границы)

Абстракция – это набор всех таких характеристик

Абстрагирование – это способ сконцентрироваться на интерфейсе (внешнее поведение объекта) не обращая внимания на реализацию.

Абстракция сущности – объект представляет собой полезную модель некой сущности предметной области

Абстракция поведения – объект состоит из обобщенного множества операций

Абстракция виртуальной машины – объект группирует операции которые либо используются более высоким уровнем абстракции либо сами используют некоторый набор операций более низкого уровня.

Произвольная абстракция –объект включает в себя набор операций не имеющих

Инкапсуляция – это свойство системы позволяющее объединить данные и методы работающие с ними в классе и скрыть детали реализации от пользователя

Инкапсуляция выполняется посредством сокрытия данных

Абстракция и инкапсуляция дополняют друг друга

Абстрагирование направлено на наблюдаемое поведение объекта дальше в методе

Наследование – это свойство системы позволяющее описать новый класс на основе уже существующего с частично или полностью заимствующейся функциональностью

Класс от которого производится наследование называется базовым или родительским. Новый класс – потомком, наследником или производным классом

Одиночное –единственный родитель

Множественное – у потомка два и более родителей

проблемы множественного наследования

1) Конфликты имен между различными суперклассами – в дух или более суперклассах определено поле иди операция с одинаковым именем

2) Повторное наследование – класс наследует двум классам а те в свою очередь порознь наследуют одному и тому же четвертому

Выход – использование виртуальных базовых классов для запрещения дублирования повторяющихся структур

Полиморфизм – это свойство системы использовать объекты с одинаковым интерфейсом без информации о типе внутренней структуре объекта.

Полиморфизм а ООП называется переопределением наследником функции члена базового класса

Чистая виртуальная функция – виртуальная функция которая объявлена со спецификатором = 0 вместо тела

virtual void Draw () const = 0;

Не имеет определения и не может быть непосредственно вызвана

Механизм работы

ели в базовом классе хоть б одна виртуальная функция то для каждого полиморфного класса создается таблица виртуальных функций – одномерный массив указателе на функции

Количество элементов в массиве равно количеству виртуальных функций в классе

Для всех полиморфных классов таблица будет содержать разные значения.

Абстрактным классом называется такой у которого есть хотя бы одна виртуальная функция

Модульность – свойство системы, которая была разложена на внутренне связанные, но слабо связанные между собой модули.

Структура модуля должна быть достаточно простой для восприятия. Реализация каждого модуля не должна зависеть от реализации других модулей. Должны быть приняты меры для облегчения процесса внесения изменений там, где они наиболее вероятны.

Типизация – это способ защититься от использования объектов одного класса вместо другого или по крайней мере управлять таким использованием.

Сохраняемость – способность объекта существовать во времени, переживая процесс, породивший его и (или) в пространстве, перемещаясь из своего начального адресного пространства.

Сериализация – процесс перевода из какой либо структуры данных в последовательность битов.

Десериализация – процесс обратный сериализации.

Любой объект существует в памяти и живет во времени

1) Промежуточные результаты вычислений выражений (ЯП)

2) Локальные переменные и объекты в блоках а также при вызове процедур и функций (ЯП)

3) Статические переменный классов а также глобальные переменный и объекты в динамической памяти (ЯП)

4) Данные сохраняемы между сеансами выполнения программы (БД)

5) Данные сохраняемы при переходе на другую версию программы (БД)

6) Данные переживают программу во времени (БД)

Параллелизм – свойство, отличающее активные объекты от пассивных.

Разделение предметной области на объекты позволит разнести общий функционал системы

Критерии оценки декомпозиции

1) Зацепление – степень глубины связей между отдельными модулями

2) Связанность – степень взаимодействия меду элементами отдельного модуля

3) Достаточность – степень необходимо для реализации логичного и эффективного поведения в классе

4) Полнота – в интерфейсной части класса должны быть учтены все характеристики абстракции

5) Примитивность – примитивными являются те операции которые требуют доступа к внутренней реализации абстракции.

Параллельное программирование

Одновременными событиями называют события если они происходят в течении одного и того же временного интервала.

Ели несколько задачи выполняются в течении одного и того же временного интервала то говорят что они выполняются параллельно

В многопроцессорной среде, если свободно достаточное количество процессоров, параллельные задачи могут выполняться в одни и те же моменты времени в течении одного и того же периода времени

в однопроцессорной среде параллельные задачи существую в одно и то же время и выполняются в течение одного и того же интервала времени

Цель параллелизма

ускорить обработку

Методы параллельного программирования – позволяют распределить работы программы между процессорами в рамках одного физического или одного виртуального компьютера

Методы распределенного программирования

позволяют распределить работу программы между двумя или более процессами прием процессы могут существовать на одном и том же компьютере или на разных

Архитектура распределенной программы

Программа делится на части – снижается общая сложность

Использование локальной сети или сети Интернет

Использование разных информационных сред

Резервирование оборудования и вычислений

Совместное использование дорогостоящих ресурсов

Схемы (модели) параллелизма

Single-Proram, Multiple-Data (SPMD)

одна программа, несколько потоков данных

все процессоры выполняют одни и те же операции но с различными данными

Multile-Proram, Multiple-Data (MPMD)

процессы выполняют различные виды работы

при этом все они вместе пытаются решить одну проблему

каждому из них выделяется свой аспект этой проблемы.

Простейшие модели распределенного программирования

Клиент – Сервер

Сервер обеспечивает опосредованный доступ к огромной базе данных дорогостоящему оборудованию или некоторой коллекции приложений

Мультиагентные распределённые системы

Автономность – агенты хотя бы частично независимы

Ограниченность представления – ни у одного из агентов нет представления о всей системе или система слишком сложна чтобы значение о ней имело практическое применение ля агента

Децентрализация – нет агентов, управляющих всей системой

Агенты могут обмениваться полученными знаниями

Может проявляться самоорганизация

Уровни параллелизма

1) Инструкций

2) Подпрограмм

3) Объектов (CORBA)

4) Приложения

Процесс проектирования параллельных системы

1) Декомпозиция

2) Связывание – связь частей между собой

3) Синхронизация – координация функционирования компонентов единой системы, порядка работа, определения критерия, когда цель достигнута

Показатели эффективности параллельного алгоритма

1) Ускорение – во сколько раз ускоряется производительность системы по сравнению с последовательными вариантам обработки

2) Эффективность – на сколько эффективно работают несколько процессоров

3) Стоимость – метод стоимость которого является пропорциональной времени выполнения наилучшего последовательного алгоритма.

Оценка максимального достижимого параллелизма

Закон Амдала

Всегда есть последовательные расчеты, которые не могут быть распараллелены

Пусть f есть доля последовательных вычислений в применяемом алгоритме обработки данных

Ускорение процесса вычислений при использовании p процессоров ограничивается величиной Sp

Закон Густавсона – Барсиса

ускорение масштабирования

Оценка может показать насколько эффективно могут быть организованы параллельные вычисления при увеличении сложности решаемых задач.

Масштабируемые алгоритм – если при росте числа процессоров он обеспечивает увеличении ускорения при сохранении постоянного уровня эффективность использование процессоров

Параллелизм в C++

Не содержит никаких синтаксических примитивов для параллелизма

Варианты:

библиотека POSIX – стандарт которые поддерживает ОС Windows

MPI – используется на компьютерах с массовым параллелизмом и кластерах рабочих станций

CORBA – стандарт для кроссплатформенного ООП. Агентно ориентированное и мультиагентное программирование

CUDA + CUDA Toolkit – архитектура параллельных вычислений от NVIDIA, позволяющая существенно увеличить вычислительную производительность благодаря использованию GPU (графических процессоров)

Процессы и потоки

1) Процесс – некоторая часть работы, создаваемая операционной системой

2) поток – часть выполняемого кода которая может быть регламентирована определенным образом. Не имеет своего адресного пространства

Процесс может иметь несколько потоков выполнения управления и выполнения

Процесс с несколькими потоками, выполняющимися параллельно, называется многопоточным.

Проблемы потоков

Трудности связи

Выбор оптимального количества процессоров

Голодание (бесконечная отсрочка) – остановка работы одной или нескольких задач на неопределенное время вследствие исполнения задач с большим или равным приоритетом и или длительным временем пробуждения или разблокировки

Гонка данных – ситуация при которой конечное состояние системы зависит от порядка и интенсивности выполнения по токов, когда поток не имеют информации друг о друге но работают с общим ресурсом и хотя бы один из них изменяет этот ресурс

Взаимоблокировки