Кросс-платформенное программирование

Лекция №1

Программирование — это искусство. Совершенство достигается тогда, когда программа, выполняющая свою функцию, занимает всего несколько строк; когда одна программа может делать то, чего не может делать другая; когда одни программы могут проникать в другие; когда программа может манипулировать с файлами такими способами, которые раньше считались невозможными. Совершенные программы, отдельные приёмы программирования, удачные алгоритмы могут быть предметом коллекционирования и почитания.

– Стивен Леви

Лектор:

к.т.н., доцент кафедры

динамики и прочности машин (к. 12)

Водка

Алексей Александрович

Рекомендованная литература

- Julian Smart, Kevin Hock, Stefan Csomor. Cross-Platform GUI Programming with wxWidgets. Prentice Hall Professional, 2005. – 744 p.
- Лафоре Роберт. Структуры данных и алгоритмы в Java. 2-е издание. – Питер, 2013. – 702 с.
- Колисниченко Д. Краткое руководство пользователя Ubuntu 10. - БХВ-Петербург, 2010. – 352 с.
- Кубенский А.А. Структуры и алгоритмы обработки данных. Объектно-ориентированный подход и реализация на C++. – БХВ-Петербург, 2004. – 464 с.
- Николас Солтер, Скотт Клепер. C++ для профессионалов. – Диалектика, Вильямс, 2006. – 912 с.

Структура курса

Содержание:

- Основы кроссплатформенного программирование на базе C++, wxWidget и Code::Blocks;
- Структуры данных (стек, очередь, связанный список, бинарное дерево);
- Стандартная библиотека шаблонов STL;
- Некоторые прикладные задачи;

Сессионный контроль

- В конце семестра – экзамен;
- Курсовая работа;
- Оценка за экзамен по рейтингу = Тест по теории + оценка за л/р + оценка за курсовую работу.

Лабораторные работы

- Лабораторная работа сданная вовремя не требует письменного отчета
- Лабораторная работа несданная вовремя требует письменного отчета
- Вовремя сданной л/р считается та, что сдана с положительной оценкой (3, 4, 5) в течении двух календарных недель с момента выдачи задания

Отчет о лабораторной работе

Национальный Технический Университет

«Харьковский Политехнический Институт»

Кафедра ДПМ

Отчет о лабораторной работе № …

По курсу «кросс-платформенное программирование»

Выполнил ст. группы …

<ФИО>

Харьков-2016

Отчет о лабораторной работе

Содержание отчета:

- Номер варианта
- Текст задания
- Рисунки форм приложения
- Текст модулей программы

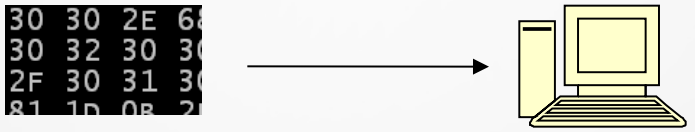
Языки программирования

общая классификация

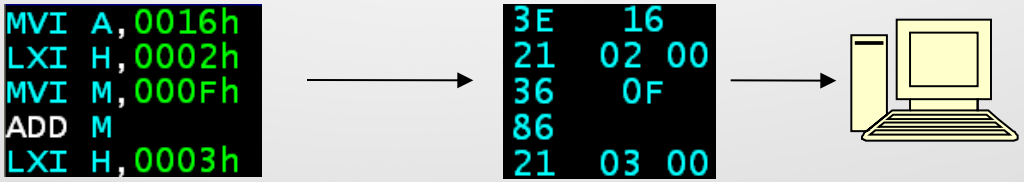
Языки программирования

Язык программирования – набор правил (лексических, синтаксических и семантических) для составления компьютерной программы.

Машинный язык



Язык ассемблера



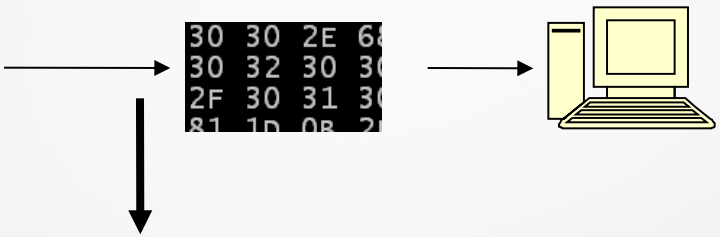
Мнемонические команды вместо машинных команд

Языки программирования высокого уровня

ЯВУ имитируют естественные языки, используя некоторые слова разговорного языка и общепринятые математические символы

FORTRAN (1954 г.) – первый ЯВУ

```
program hello
print *, "Hello, world!"
real,dimension(:,:) :: v
allocate(v(-2:2,0:10))
end
```



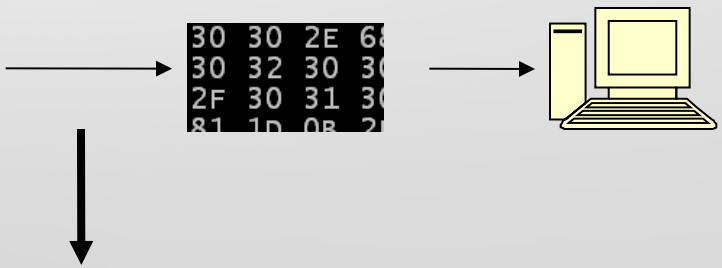
Структурное программирование

Управляющие структуры, подпрограммы (функции, процедуры), рекурсия, локальные переменные, отсутствие GOTO

Алгол(1958), Паскаль(1970), Си(1972).

```
begin
k := process(12,
writeln(k);
end;
```

```
function process(
var s: real;
begin
s := m/3 + n;
```



Объектно-ориентированное программирование (ООП)

Классы, объекты, инкапсуляция, полиморфизм, наследование
Object Pascal, C++, Java, C#, … .

ООП позволяет оптимально организовывать программы, разбивая проблему на составные части, и работая с каждой по отдельности.

Класс представляет собой тип данных, объединяющий поля (свойства) и методы (функции).

Класс «Человек»

Поле «Имя»

Метод «Получить имя»

Метод «Отправить сообщение»

Экземпляр (объект) класса «Человек»

Поле «Имя» = Вася

Метод «Получить имя»

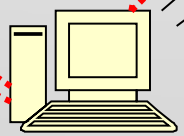
Метод «Отправить сообщение»

Экземпляр (объект) класса «Человек»

Поле «Имя» = Петя

Метод «Получить имя»

Метод «Отправить сообщение»



Объектно-ориентированное программирование (ООП)

Инкапсуляция –

Наследование –

Полиморфизм –

Объектно-ориентированное программирование (ООП)

Инкапсуляция – объединение данных и методов для работы с ними в один объект. Инкапсуляция также реализует сокрытие данных от внешнего воздействия, что защищает их от случайного изменения.

Наследование – это способ повторного использования программного обеспечения, при котором новые классы создаются из уже существующих классов путем заимствования их атрибутов и функций и обогащения этими возможностями новых классов.

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

Класс «Человек»

Поле «Имя»

Метод «Получить имя»

Метод «Отправить сообщение»

Свободное ПО и закрытое ПО. GNU GPL.

GNU General Public License (Универсальная общедоступная лицензия GNU или Открытое лицензионное соглашение GNU) — популярная лицензия на свободное программное обеспечение, созданная в рамках проекта GNU в 1988 г.

GNU - (GNU is Not Unix — «GNU — это не Unix») — проект по созданию свободной UNIX-подобной операционной системы, начатый Ричардом Столлмэном в 1983 году.

ОС GNU/Linux = системные утилиты проекта GNU + ядро Linux.

Свободное ПО и закрытое ПО. GNU GPL.

Права (свободы) для пользователя компьютерной программы:

- доступ к исходному коду;

- свобода изучения того, как программа работает, и её модификации;

- свобода распространения копий;

- свобода улучшения программы, и выпуска улучшений в публичный доступ.

Пользователи производных программ получат вышеперечисленные права («copyleft» - принцип «наследования» прав)

Анализ сложности и эффективности алгоритмов и структур данных

В процессе решения прикладных задач выбор подходящего алгоритма вызывает определенные трудности. Алгоритм должен удовлетворять следующим противоречащим друг другу требованиям:

1. быть простым для понимания, перевода в программный код и отладки;
2. эффективно использовать вычислительные ресурсы и выполняться как можно быстрее.

Виды эффективности алгоритмов

Эффективность алгоритма:

- ***Временная*** (индикатор скорости работы алгоритма)
- ***Пространственная*** (сколько для алгоритма требуется оперативной памяти)

В основном анализируется временная эффективность.

Оценка размера входных данных

Временная эффективность (далее просто **эффективность**) напрямую зависит от размера входных данных.

Эффективность можно задать функцией от некоторого параметра, связанного с размером входных данных.

Пример:

- Эффективность алгоритма решения задачи сортировки списка зависит от размера списка

Оценка размера входных данных

Для некоторых задач можно брать разные параметры входных данных.

Пример:

- Задача перемножение двух матриц размером n на *n*.

1-ый вариант параметра - это порядок матрицы n

2-ой вариант - число *N* =n*n элементов в матрице

Оценка размера входных данных

В случае, если на вход алгоритму подается целое число *n*, принято оценивать размер входных данных по количеству битов *b* в двоичном представлении числа *n*.

$$b = \log_2 n + 1$$

Например, для алгоритма возведения целого числа *n* в квадрат значение эффективности одинаково для чисел 9 и 14 (т.к. число битов двоичного представления этих чисел одно и то же).

Единицы измерения времени

В качестве единицы измерения времени выполнения алгоритма принято использовать не секунды, минуты и пр., а количество операций, выполняемых алгоритмом.

При этом считаются не все операции, а лишь ***основные*** (которые вносят наибольший вклад в общее время выполнения алгоритма).

При анализе эффективности алгоритма необходимо определить, какие операции в нем основные.

Единицы измерения времени

Примеры

В большинстве алгоритмов сортировки основной операцией является сравнение.

В алгоритме умножения матриц используются операции сложения и умножения, основной является умножения, т.к. на компьютерах оно выполняется дольше, чем сложение.

Единицы измерения времени

Пусть *c* – время выполнения основной операции алгоритма, а *C*(*n*) – количество раз, которые эта операция должна быть выполнена при работе алгоритма.

Время выполнения программы *T*(*n*) определяется по формуле $T(n) \approx c \cdot C(n)$

Данная формула позволяет определить, на сколько изменится время выполнения алгоритма при изменении размера входных данных.

Единицы измерения времени

Пусть $C(n) = \frac{1}{2} n^2$

Если удвоить размер входных данных, то время выполнения алгоритма увеличится в 4 раза, т.к.

$$\frac{T(2n)}{T(n)} \approx \frac{c \cdot C(2n)}{c \cdot C(n)} \approx \frac{\frac{1}{2}(2n)^2}{\frac{1}{2} n^2} = 4$$

Заметим, что ответ не зависит от *c*, поэтому при анализе эффективности алгоритмов сосредотачиваются на оценке ***порядка роста*** количества основных операций с точностью до постоянного сомножителя.

Сложность алгоритмов в разных случаях

В большом количестве алгоритмов время выполнения зависит не только от размера входных данных, но и от особенностей конкретных входных данных.

Пример.

Задача последовательного поиска (поиск заданного элемента *k* в списке длиной *n*). В наихудшем случае (когда заданного элемента в списке нет) будет выполнено *n* операций сравнения. В наилучшем – будет выполнено 1 сравнение.

Сложность алгоритмов в разных случаях

Сложность алгоритма в наихудшем (наилучшем) случае – сложность для таких входных данных, на которых время работы алгоритма будет наибольшим (наименьшим).

Упражнения

В ящике хранится 22 перчатки: 5 пар красных, 4 пары желтых и 2 пары зеленых. Предположим, что вы выбираете их в темноте наугад и можете проверить, что именно вы выбрали, только после того, как выбор сделан. Чему равно минимальное количество перчаток, которое надо взять из ящика, чтобы получить как минимум одну пару перчаток (т.е. левая + правая) одинакового цвета?

а) Дайте ответ на этот вопрос для наилучшего и наихудшего случая.

б) Дайте ответ для наилучшего и наихудшего случая для варианта, в котором нужно получить как минимум две перчатки одинакового цвета (не обязательно пару)

Классы сложности

Факт $\lim_{n->\infty} \frac{f(n)}{g(n)} = C$ где *C* - константа

обозначается так: $f(x) = O(g(x))$, это значит что функции f(x) и g(x) имеют ***один и тот же порядок роста.***

Класс	Название
1	Константная
n	Линейная
n^2	Квадратичная
n^3	Кубическая
2^n	Экспоненциальная
$n!$	Факториальная

Запись $f(x) = O(g(x))$ означает, что $f(x)$ относится к классу сложности $g(x)$, а также, что $f(x)$ имеет соответствующий порядок роста (например, линейный)

Классы сложности

На практике при поиске класса сложности можно отбрасывать слагаемые, которые представляют функции меньшего порядка роста.

Примеры:

а) $5n + 3 = O(n)$, т.к. $\lim\limits_{n->\infty} \frac{5 \cdot n + 3}{n} = \lim\limits_{n->\infty} \frac{5 \cdot n}{n} + \lim\limits_{n->\infty} \frac{3}{n} = 5 \cdot \lim\limits_{n->\infty} \frac{n}{n} = 5$

б) $\frac{n \cdot (n-1)}{2} = O(n^2)$, т.к.

$$\lim_{n->\infty} \frac{n \cdot (n-1)/2}{n^2} = \frac{1}{2} \cdot \lim_{n->\infty} \frac{n^2 - n}{n^2} = \frac{1}{2} \cdot \lim_{n->\infty} \left(1 - \frac{1}{n}\right) = \frac{1}{2}$$

Упражнения

Для каждой из приведенных ниже функций укажите порядок роста и класс сложности.

а) $\left(n^2+1\right)^{10}$

б) $\sqrt{10n^2+7n+3}$

в) $2^{n+1}+3^{n-1}$

Анализ нерекурсивных алгоритмов

Общий план анализа сложности нерекурсивных алгоритмов:

1. Выберите параметр (или параметры), по которому будет оцениваться размер входных данных алгоритма.

2. Определите основную операцию

3. Определите сложность алгоритма в наихудшем случае

   3.1. Запишите сумму, выражающую количество выполняемых основных операций алгоритма

   3.2. Упростите формулу (используя правила суммирования), и определите, к какому классу сложности относится полученная функция

4. Определите сложность в наилучшем случае (если она не совпадает с наихудшей)

Анализ нерекурсивных алгоритмов

Важные правила и формулы суммирования:

$$\sum_{i=k}^{n} c \cdot a_i = c \cdot \sum_{i=k}^{n} a_i$$

$$\sum_{i=k}^{n} \left(a_i \pm b_I\right) = \sum_{i=k}^{n} a_i \pm \sum_{i=k}^{n} b_i$$

$$\sum_{i=k}^{n} 1 = n - k + 1$$

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

Анализ нерекурсивных алгоритмов

Пример 1. Задача поиска наибольшего элемента в массиве из *n* чисел.

```
maxval = a[0];
for (int i = 1; i < n; i++)
  if ( a[i] > maxval )
        maxval = a[i];
```

Размер входных данных = *n*, основная операция = сравнение, сложность в наихудшем и наилучшем случае одинаковая.

$$C(n) = \sum_{i=1}^{n-1} 1 = (n-1) - 1 + 1 = n - 1 = O(n)$$

Ответ: сложность алгоритма линейная

Анализ нерекурсивных алгоритмов

Пример 2. Задача проверки уникальности элементов в массиве из *n* чисел.

```
for (int i = 0; i<= n-2; i++)
  for (int j = i + 1; j<=n-1; j++)
        if ( a[i] == a[j] )
              return false;
return true;
```

Основная операция = сравнение.

Размер входных данных = *n*

Сложность в наихудшем и наилучшем случае различная

Анализ нерекурсивных алгоритмов

$$C_{worst}(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1 = \sum_{i=0}^{n-2} ((n-1)-(i+1)+1) =$$

$$= \sum_{i=0}^{n-2} (n-1-i) = \sum_{i=0}^{n-2} (n-1) - \sum_{i=0}^{n-2} i =$$

$$= (n-1) \sum_{i=0}^{n-2} 1 - \frac{(n-2)(n-1)}{2} =$$

$$= (n-1)^2 - \frac{(n-2)(n-1)}{2} = \frac{(n-1)n}{2} = O(n^2)$$

$$C_{best}(n) = 1 = O(1)$$

Ответ: сложность алгоритма в наихудшем случае квадратичная, в наилучшем - константная

Анализ нерекурсивных алгоритмов

Пример **3**. Задача умножения двух квадратных матриц размером *n* на *n*.

```
for (int i=0; i < n; i++)
   for (int j=0; j < n; j++) {
   C[i][j] = 0;
    for (int k=0; k < n; k++)
      C[i][j] = C[i][j] + A[i][k] * B[k][j];
  }
```

Основная операция = умножение.

Размер входных данных = *n*

Анализ нерекурсивных алгоритмов

Определите сложность в наихудшем и наилучшем случае

$$C(n) = \sum_{i=0}^{n} \sum_{j=0}^{n} \sum_{k=0}^{n} 1 = n^3$$

Сложность в наихудшем и наилучшем случае одинаковая.

Ответ: сложность алгоритма кубическая

Анализ сложности и эффективности алгоритмов и структур данных

Для примера приведем числа, иллюстрирующие скорость роста для нескольких функций, которые часто используются при оценке временной сложности алгоритмов

n	log(n)	nlog(n)	n^2
1	0	0	1
16	4	64	256
256	8	2048	65536
4096	12	49152	16777216
65536	16	1048576	4,29E+09
1048576	20	20971520	1,10E+12
16777216	24	402653184	2,81E+14

Если считать, что числа соответствуют микросекундам, то для задачи с 1048576 элементами алгоритму со временем работы T(log n) потребуется 20 микросекунд, а алгоритму со временем работы $T(n^2)$ - более 12 дней.