**С/C++**

"Ох волна моя, волна, ты как С++ мощна"

Почти Пушкин

Касательно мощности C/С++.Да, действительно это очень мощные языки ,но их мощность заключается в их средствах низкого уровня,а они при разработке прикладных программ часто неприменимы.



Мощность - это, конечно, хорошо,но часто опасно.

Пример: #define int long синтаксически верно, но может привести к катастрафическим последствиям (хотя, к примеру, при переносе программ с одной платформу на другую это может понадобиться). Или оператор goto. То, что он не включен в Java, многие считают очень большим благом. Goto - это мощная возможность, но, как показал еще в семидесятых годах Дейкстра, мягко говоря, "не рекомендованная" к использованию. Тем не менее это возможность, и она может быть не лишней. Я считаю, что подобные низкоуровневые конструкции должны присутствовать, но ими не следует злоупотреблять. В тех редких случаях, когда лично я его использовал, это была попытка развития плохоспроектированной программы.

"Чтобы сделать карьеру программиста за границей нужно знать два языка: английский и C++"

Российский аспирант, побывавший в Вене.

"С точки зрения теоретического программирования язык си - это Фортран 80 - ых. (Против такого уничижительного определения не возражал и автор языка Си - Д. Ритчи). Этот язык, сочетающий в себе многие преимущества языка высокого уровня и ассемблера, дал программистам, по образному выражению некоторых, педаль газа, но заблокировал педаль тормоза. На Си компьютер может "мчаться" быстро, но рискованно. То есть Си, насаждая ссылочно - ассемблерное программирование, как бы имеет вектор в сторону, противоположную той, которая определяется теорией и методологией языков программирования.

Поэтому и Си++ - это довольно странное сочетание некоторых черт ООП (здесь и далее - объектно - ориентированное программирование) и процедурного программирования"

К.Т.Н. Соловьев А.Е.

"Cи++ представляет собой интересный эксперимент по адаптации возможностей объектной технологии к традиционному языку программирования. ... поскольку Си++ - язык, требующий весьма интенсивной критики. Он представляет собой не слишком удачную реализацию объектно - ориентированной технологии, и поэтому его недостатки просто необходимо подвергать критическому анализу... Си++ приносит всем колоссальное разочарование - он вобрал в себя все плохие и старые средства, а также привнес в объектную технологию абсолютно ненужную сложность. Зачем нужен язык, насквозь пропитанный низкоуровневыми конструкциями? ".

Ян Джойнер

Далее по тексту "Наиболее верный путь к успеху - использование чистого объектно - ориентированного языка, обладающего интерфейсом с языком Cи или с другими низкоуровневыми языками. Это обеспечит хорошие поддержку и сопровождение, переносимость и качество", что в принципе можно наблюдать на примере любого средства 4GL(того же Delphi) поддерживающего стандартные средства взаимодействия Windows такие как Dll,Dde, Ole. Простейший пример - обращение из Delphi программы к Win API(ведь Windows написан на Си(9X) и C++(NT)).



Изначально созданный для разработки ОС Unix, язык C получил широкое распространение. Это наложило очень сильный отпечаток на язык.

Как возник язык С? Вначале был CPL: он был создан в середине 60 - х годов.

Язык не получил широкого распространения, но в процессе его создания появилась масса идей. Язык был беcтиповый, как и положено ассемблеру (ещё PDP 7).

Новая упрощенная версия языка называлась Basic CPL или BCPL. На нем была реализована MULTICS. Но в результате ее размеров и раздутости ей потребовалась замена.

Часть идей из MULTICS была взята разработчиками Unix. К примеру, иерархическая файловая система. Замечу, что в прямом предшественнике DOS-CP/M её не было, как и в первой версии DOS.



Доработали язык - получился B.

Разработали ОС - получилась Unix

Си разрабатывался как Кроссплатформенный ассемблер PDP 11 (многие опытные программисты знают его по отечественным аналогам), является некоторой смесью ассемблера с Паскалем, этим обедняются многие его особенности.

К примеру:

Отсутствует возможность напрямую работать с данными, не поддерживающимися процессором PDP (множества, строки, etc).

Присутствие явно PDP - ориентированных конструкций:

Функция poly, сводящаяся к аналогичной команде ассемблера. Вычисление же полиномов не является задачей первостепенной важности языка.

Постинскремент и прединскремент. На x86, к примеру, это ничего не дает. И A=B++ и A:=B; inc(b); приведут к командам ассемблера MOV a,b; inc b (естественно, непосредственно в коде это будут не имена переменных, а, к примеру, регистры).

" .. что аномалии существующих вычислительных машин старательно воспроизводятся в языках программирования, причем это происходит в ущерб интеллектуальной управляемости программ, выражаемых на таком языке.." Э.Дейкстра. "Дисциплина программирования"



Вирт писал что : "...язык должен определяться в терминах математических, абстрактных концепций И только, если язык удовлетворяет этому критерию, он может считаться высокоуровневым."

Изменяющиеся от платформы к платформе размеры типов данных.

Строки, оканчивающиеся нулем, позаимствованные из ассемблера (директива .asciz ассемблера PDP)

Отсутствие автоматической либо обязательной инициализации переменных. Плюс такого подхода - большая оптимальность кода, минус - необходимость производить инициализацию вручную. Отсутствие автоматической инициализации является потенциальной возможностью для разработки ПО, содержащего неустойчивые ошибки (в идеале компилятор должен отслеживать не производится ли чтение не инициализированной переменной и в случае, если проводится, обнулять ее).

Наличие множества "подводных камней"

scanf("%d %d",&n,&ar[n]);

Вы думаете, что будет введено целое n и n - ный элемент масива ar ?

Это очень маловероятно. Скорее всего, все аргументы scanf будут вычислены до того, как операция будет вызвана на выполнение. Подобных побочных эффектов в языке масса.

Другая "особенность"

char

str[50]="qwertyuio";

int a=3;str[++a]=str[++a]=' ';

cout<<str<<"\n";

str[a++]=str[a++]=' ';

cout<<str<<"\n";

Результат:

qwe tyuio

qwert uio

По логике вещей должны быть добавлены два пробела. Но у C своя логика, так что это не ошибка компилятора, а особенность языка.

int i=0,ar[2]={10,11};

i=ar[i++];// А кто сразу скажет чему равно значение i

Хотите еще ? Формат вывода зависящей от типа:

short int x=55;

printf("%d\n",x);

Если заменить short int на longt int, то придется менять и printf("%d\n",x) на printf("%D\n",x)

Пример: предположим, что вместо i<=100 разработчик написал i=100 при синтаксисе C/C++

for (i=0; i=100; i++)

Цикл будет выполняться вечно (вместо 101 раза) т.к.

Отсутствует логический тип данных.

Выраженние i=100 равно 100(т.е. по не ноль - истина).

Поскольку C включает в себя элементы не только процедурного, но и функционального программирования такая конструкция вполне правильна и логична.

По мере развития в C включалось все больше возможностей Изначально язык не имел средств даже для описания констант. Когда Си стал применяться для решения серьезных задач, к нему добавили так называемые "директивы препроцессора", такие, как #define и #include.



С помощью define стали определять константы и inline подпрограммы,

А с помощью #include был реализован, хоть и примитивный, механизм модульности. Популярность функционального программирования тоже сыграла свою роль. В языке появились конструкции из функциональных языков.

Конечно, у этой особенности есть и более достойное применение

if (сh=getchar()!=ESC)

{..}

Обобщу, что такие средств хоть и удобны в использовании и позволяют писать разработчику "красивые" программы, не способствуют безопасности этих программ и совсем не считаются простотой языка. Так что "Красиво"- не всегда хорошо.



Из минусов также следует отметить не слишком читабельный синтаксис. Подумайте, что больше говорит end loop в АДЕ или "}" в C. Конечно, краткость - это хорошо, но платить за нее такую цену....

Примеры:

Паскаль:

if Screen.Forms[I] is FormClass then begin

C++:

if (dynamic\_cast<FormClass\*>(Screen - >Forms[I])){

A=(!CL&&!RC)?0 : (!LC?RC:LC)//"Очень понятное выражение"

\*++\* agrv //"Еще одно очень понятное выражение, при том синтаксически верное"

"Интуитивно понятный" синтаксис прекрасно подчеркивает следующий пример:

int i=5;

int \*

const p3=&i;//Указатель константу

const int \*

p3=&i;//Указатель на константу

i=5;//Правильно

\*p3=5; Неверно! указатель на константу измененную в предыдущей строке.

char (\*(\*x2 ())[]) () //Срочно позовите криптоаналитика !!!

Или работа с перечислениями:

enum modes { LASTMODE , BW40 , C40, BW80, C80, MONO

} ;

..

modes

e1=C80,e2;

e1=e2\*3;

//Очень осмысленный оператор на "ЯПВУ".

//Ведь для "ЯПВУ" нет разницы, что int, что enum, что

bool

А что может значить, по Вашему мнению, команда a=24[ar]; ?

При условии, что int ar[50]; int a; она полностью эквивалентна a:=ar[24];

Как совершенно справедливо замечают поклонники C/C++ эти языки позволяют писать чрезвычайно краткие и выразительные программы. На счет краткости - безусловно.

А вот к какой выразительности может привести краткость, я сейчас покажу:

#include <stdio.h>

#define Q r=R[\*p++ - '0'];while(

#define B ;break;case

char\*s="Qjou!s\\311^ - g\\311^ - n\\311^ - c\\::^ - q - ma%mO1JBHm%BQ - aP1J[O1HB%[Q<nbj\

o)\*|gps)<<\*txjudi)m\*|aQdbtf!::::;sfuvso<aQefgbvmu;aQ<m,,a%CQ<csfbla%bQ<aN2!Q\

\ndbtf!aP2Q;m>aP2Q<a%!D12J!JGJHJOJQJFJSJJJMHS%HD12D12N3!N4\nJUJT%UQm>aP4HC%T\

Qs\\q,,^>m,2<m>aP4HC%SD12N1\nJNQm>s\\..q^aHC%NHb%GN1!D32P3%RN1UP1D12JPQUaP1H\

R%PN4\nQ<g\\(aP3Q(^>aP2Q,2<n\\(aP3Q(^>aP4Hb%OD12D12N2!N3\nJVP3Q,,<jg)aP3Q=>n\

\\(aP3Q(^\*m>g\\(aP3Q(^<fmtf!m,,aHC%QN1!N1\nJ#Qqsjoug)#&e]o# - aP1Q\*aHb%#Qqvut)\

aP1Q\*aHb%FN1\nQm>::::aHC%VP3Q>bupj)hfut)c\*\*aHb%JD12JON1!Qjg)a%LN1UP1D12JIQUa\

P1HL%IQ\*m>aN2!N2\nP2Q<fmtf!m,,aHC%MN1!N2>P2Q>aN2\nP2Hbdd!b/d";int k;char R[4][99]

;main(c,v)char\*\*v;{char\*p,\*r,\*q;for(q=s;\*q;q++)\*q>' '&&(\*q) - - ;{FILE\*i=fopen(v

[1],"r"),\*o=fopen(q - 3,"w");for(p=s;;p++)switch(\*p++){B'M':Q(k=fgetc(i))!=EOF

&&k!=\*p)\*r++=k;if(k==EOF){fputs("}}\n",o);fclose(o);return system(q - 6);}\*r=0

B'P':while(\*p!='`')fputc(\*p++,o)B'O':Q\*r)fputc(\*r++,o);p - - B'C':k=0;Q k<\*p - '0'

)(\*r++=fgetc(i),k++);\*r=0 B'I':k= \*p;if(\*\*R==k)goto G B'G':k= \*p;G:p=s;while(

\*p!='$'||p[1]!= k)p++;p++B'N':R[\*p - '0'][0]++;}}}

Эта программа всего в 17 строчках текстового режима VGA(25X80) представляет собой полнофункциональный интерпретатор языка Basic, который поддерживает:

Переменные(имена от A до Z), которые инициализируются нулевыми значениями при запуске.

Цикл FOR var = exp TO exp ..NEXT var

Подпрограммы GOSUB exp и RETURN

Естественно, оператор GOTO(какой же Basic без GOTO)

Условия IF exp THEN exp

Комментарий REM any text

Оператор конец программы END

Присвоение var = exp

Ввод INPUT variable

И вывод PRINT string PRINT exp

Есть ли читатели, которые по - прежнему считают, что краткость - это всегда очень хорошо. А читабельность конструкций языка факт второстепенный ?

В части ясности синтаксиса антиподом C/С++ является язык АДА. Вместо бесконечных "}" пишется END LOOP, END IF, END CASE или END ИМЯ\_ПОДПРОГРАММЫ/ПАКЕТА. Я уже не говорю, насколько "Переменная ТИПА целый" читабельней чем "цел Переменная". Логично предположить, что платой за краткость является читабельность программ, или платой за читабельность программ является отсутствие краткости.

Хотя, что краткость в синтаксисе является сестрой таланта является крайне сомнительным. Отсутствие избыточности на уровне синтаксиса приходится компенсировать обильными комментариями (если, конечно, разработчик хочет иметь возможность разобраться в этом коде через пару месяцев).

Отмечу, что подобный синтаксис Microsoft положила в основу языка Visual Basic.

Ну и, конечно, отсутствие строгого контроля типов(хотя в этом есть и обратная сторона - в Pascal'е Вам придется для того, чтобы иметь возможность "прострелить себе ногу", явно указать на свое желание компилятору.

Распространенности C послужила:

Распространенность Unix.

Хорошее качество генерируемого кода. Особенно на PDP.

Достаточная "мощь" языка.

Сравнительная простота реализации компиляторов.

Краткость синтаксиса и его "красота" любима многими С/C++ - программистами, многие считают их чуть ли не основными преимуществами языка (то есть возможность написать ar[c++] вместо ar[c];inc(c) оказала большое влияние на популярность языка).

Широкие возможности разработки системного ПО(к примеру, адресная арифметика , битовые поля ). В этой низкоуровневости для многих разработчиков даже прикладного ПО есть некая прелесть, привлекающая к языку.

Справедливости ради скажу, что именно такие низкоуровневые средства часто делают C/C++ более предпочтительным для задач системного программирования по сравнению с Паскалем(Правда, уже в Modula 2 появилась адресная арифметика).



Позже сам Керниган советовал разработчикам не использовать битовые поля, так как они "слишком платформозависимы".

Итог - C хороший язык системного программирования, кроссассемблер, позволяющий очень эффективно использовать PDP, не очень хорошо подходит для разработки прикладного ПО. Замечу также, что некоторые разработчики "забавы ради" занижают уровень выбираемого для разработки языка(смотрите главу о человеческом факторе).

Перед разработчиками C стояла задача создать язык, обладающий преимуществами и ассемблера и ЯПВУ. С задачей они, в принципе, справились хорошо: правда, с плюсами оных были добавлены и их минусы.

C да и C++ не были и не будут безопасными языками, поскольку один из приоритетов этих языков - эффективность. Поэтому Страуструп не включил в C++ динамическую проверку типов. Как гласит базовый принцип ЯП, эффективность и безопасность несовместимы. Да и потери в эффективности не столь значительны для большинства задач.

Да и потеря не столь велика. Я провел небольшой эксперимент со следующей программой и её аналогом на Pacal.

#include<stdlib.h>

#include<stdio.h>

#include<time.h>

void

main()

{ unsigned long int i;

int

ar[10000];

time\_t t,t2;

t = time(NULL);

randomize();

//Многократное присвоение случайному элементу случайного значения

for (i=1;i<429496;i++)

ar[random(1000)]=random (32767);

//Многократное заполнение

массива числом 2

for (i=1;i<429496;i++)

ar[i%(1000)]=2;

t2 = time(NULL);

printf("Время выполнения= %d",t2 - t);

}

Как видно, программа очень интенсивно работает с массивом и контроль за выход границ массива должен, вероятно, замедлить работу. Ведь на каждое присвоение приходится проверка индекса массива.

Выводы могут оказаться для кого - то неожиданными:

Версия на C не показала более высокого быстродействия по сравнению с Pascal версией.

Быстродействия Pascal версии при отключении всех проверок увеличивалось на 7 - 8%.

Что вполне логично, т.к. проверка у процессоров 80x86 реализуются на аппаратном уровне. А прирост в 7 - 8% может дать использование компилятора с хорошей оптимизацией.

По данным Дмитрия Беленко, основанным на проводимом им эксперименте, разница между Delphi и C++ Builder даже вычислительных задачах составляет 4 - 5% .

Замечу что существует и с специальный проект Cyclone. Его разработчики - Корнельский университет и AT&T. Язык фактически представляет Си с проверкой потенциально опасных ситуаций, таких как переполнение буфера. В дистрибутив входит программа преобразования программ из Си в Cyclone которая может отыскать и найти потенциально опасные места программы. Основная цель разработчиков - создать язык пригодный для программирования безопасных приложений.

После появления и распространения ООП львиная доля С программистов перешли на С++ . В этом(и большом обьеме ПО уже написанном на С) я вижу одну из основных причин его распространенности.

Отмечу что и C++ повлиял на C. К примеру enum С получил от С++, и при программировании на С он практически не используется, однако было отмечено, что С содержит такую конструкцию, поскольку она есть в ныне действующем стандарте языка.



В языке появились:

классы (classes);

шаблоны (templates);

пространства имен (namespaces);

перегрузка (overload);

потоки (streams);

исключения (exception)

ООП и современные компиляторы частично сгладили недостатки С. Появились классы для работы с такими типами данных, как строки и множества. В сомнительных(смотрите пример выше) ситуациях компилятор(далеко не любой) может выдать подсказку. Для лучшей читабельности IDE автоматически выровняет исходный текст(Первоначально этим занималась команда indent, которая существует и сейчас во многих клонах UNIX(в т.ч. Linux)). Специальные средства, к примеру, NuMega BoundsChecker позволяют отслеживать наиболее вероятные ошибки, такие, как выход за границы массива или утечка памяти. Программа lint(входит в Unix'ы) занимается исключительно ревизией исходного текста программы по поводу контроля типов.

Низкая скорость компиляции в результате отсутствие полноценного механизма модульности часто компенсируется механизмом предварительной компиляции заголовочных файлов.

"Software engineering - искусство программирования без умения это делать"

Эдсгар Дейкстра (Edsger Dijkstra)

"Использования C++ для разработки прикладного ПО и вспомогательных средств для нейтрализации недостатков языка подобно использованию решета для переноски воды и специальных средств для замыкания дыр в решете."

Автор

Хотя С++ частично сгладил недостатки(просто прикрыв их своими средствами, сами недостатки никуда не делись) к нему он прибавил и свои:

Главный - черезмерная сложность и размер языка(это роднит его с Адой, хотя сложность Ады вызвана разнообразием средств, а не их дублированием). Даже реализация ООП подвергается интенсивной критике. Я для примера рассмотрю множественное наследование. Сам вопрос о необходимости включения множественного наследования в язык является достаточно спорным. В частности стоит проблема наличия одноименных методов у родителей класса. Возникает вопрос: что вызывать? Первый попавшийся ? Или все по очереди ? На практике это решается написанием у нового класса метода, который непосредственно реализует нужную логику, правда тогда возникает вопрос о адекватности самого множественного наследования. Ведь его смысл в автоматическом вызове методов (и доступе к полям) классов предков. Если же разработчику приходится указывать конкретный класс - предок, то множественное наследование в значительной степени теряет смысл. Остаются лишь такие частности как возможность обращения к protected элементов классов предков, но соответствующая модификация программы это не проблема. Реализацию множественного наследования тоже нельзя назвать тривиальным делом, она значительно сложней реализации "единичного наследования".

Большая избыточность средств - т.е. для решения одной задачи я могу использовать большое количество идентичных средств.

Кстати, многие средства языка, которые, по мнению его поклонников, появились в C++, были заимствованы из других языков. К примеру, аналог шаблонов - пакеты - были в АДЕ до создания C++. Обработка исключительных ситуаций присутствует в АДЕ и даже в PL/1.Перезагрузка операций также присутствовала в АДЕ.

Огромное разнообразие средств, которые дублируют друг друга , нравится многим поклонникам C++.Они считают, что это делает язык более удобным для использования. Считать так- это их право. Вероятно, для кого-то это действительно так.



Приведу авторитетное мнение Кена Томпсона (одного из разработчиков C и Unix):

"При работе с языками C++ и Java у меня возникало определенное беспокойство, когда я просил сделать что - либо, а в ответ получал: "Хорошо, вы делаете это так - то или можете сделать так - то". Совершенно очевидно, если вы в состоянии сделать что - то столь разнообразными способами и все эти способы более не менее эквивалентны, значит в системе заложено слишком много возможностей."

C тоже избыточен. Пример цикла без тела: for(a=1,b=70;a<50;ar[a++]=a+b,b--).По сути здесь for выполняет роль while.



Как заметил У. Вульф, "Факт наличия тех или иных возможностей в некотором ЯП не может служить критерием для оценки этого языка. Большой набор хороших самих по себе возможностей, собранных воедино без общей идеи, без определенного единообразия и без элегантности, не приводит к появлению хорошего ЯП."

Средства поддержки совместимости с С.С одной стороны это большой плюс(очень много ПО написано на C), с другой, для обьектно - ориентированного программирования они не нужны.

Пример: введение классов(при их использовании) делает ненужным записи(struct) и вариантные записи(union) , тем не менее по соображениям совместимости они присутствуют.

С++ является надмножеством C , т.е. некая высокоуровневая оболочка к языку системного программирования. Но база для этой оболочки все та же - C.

К примеру, добавлены классы для работы со строками, но сами строки реализованы через указатели. Конечно, здесь есть и плюс-широта средств языка. Но она не компенсирует получившуюся громоздкость.

Макросредства ассемблеров тоже являются высокоуровневой оболочкой и позволяют использовать конструкции, характерные для ЯПВУ (типа циклов) в программах на ассемблере. Замечу, что и на макросредствах ассемблера без использования непосредственно команд процессора можно создавать достаточно сложные системы. Но это не делает Ассемблер языком высокого уровня.

"Писать плохие программы на С++ значительно легче, чем хорошие."

Общепризнанный факт.

Громоздкость С++ Бьярн Страуструп объясняет следующим образом:

"Следовательно, универсальный язык программирования должен поддерживать различные способы мышления и стили программирования. Это разнообразие есть следствие, как разнородности решаемых задач, так и многообразия путей их решения. C++ поддерживает широкий спектр стилей программирования и поэтому более заслуживает название мультипарадигменного, нежели объектно - ориентированного языка...

Мне представляется, что языки, опирающиеся на какую - либо одну парадигму,ограничивают свободу программиста. Платой за простоту оказывается смирительная рубашка, сковывающая интеллект разработчика.."

Т.е. получается, если разные разработчики используют разные стили программирования,то полную поддержку всех стилей нужно включать в язык.

Продолжая идею, замечу - для решения разных задач необходимы разные алгоритмы. Поэтому в STL необходимо включить алгоритмы решения задач из всех областей человеческой деятельности, где используется язык C++ !!!

А в разных ОС интерфейсы разработчика существенно различаются(сравните Win32 API и POSIX).Это тоже необходимо отобразить в языке.!!!

Страуструп недавно в интервью высказался за включение в язык средств разработки баз данных и интерфейса пользователя (как в Java). Конечно, включить все, что бывает в ЯП в один язык - идея хорошая, но Вам она ничего не напоминает ? Вспомните судьбу "грандиознейшего проекта всех времен и народов - языка АДА". Вопрос о границе, где кончаются средства языка и начинаются другие средства, затронут в разделе "О маленьких и больших языках".

И последнее - что бы ни писал Страуструп, факт остается фактом. C++ полностью практически никогда и никем не применяется. Используются лишь ограниченные подмножества языка, поддерживающие определенный стиль разработки. Как он сам заметил, "Для того, чтобы писать хорошие программы на C++, необязательно знать его полностью"

В принципе, люди от естественных языков берут лишь очень небольшую часть. Но в программировании такой подход может иметь большие негативные последствия. К примеру, может появиться необходимость модификации программного кода написанного одним разработчиком, использующим 20 % языка другим, использующим другие 20 % языка(другую часть).



С++ вобрал в себя средства для поддержки очень широкого спектра технологий разработки и стилей программирования. На нем можно разрабатывать с почти одинаковой [не]успешностью, используя и линейное , и обьектно - ориентированное программирование.

Линейное программирование - архаичный способ разработки программ без использования таких структур языка, как циклы. Зато интенсивно используется оператор GO TO. Был вытеснен структурным программированием.



Собственной идеологии язык не имеет. Поддерживается множество парадигм. Разработчик должен выбрать то, что необходимо лично ему, а об остальном он может спокойно забыть.

Даже уровень языка однозначно определить невозможно. Он поддерживает как низкоуровневые структуры данных(битовые поля) и методы программирования (адресная арифметика), так и высокоуровневые структуры данных(<map> - ассоциативный массив) и методы программирования(ООП).

Всё это делает язык крайне универсальным (и даже уникальным), пригодным для решения очень широкого круга задач. Другой вопрос, что такая универсальность для львиной доли практических задач не нужна. Да и вынести низкоуровневый код, в отдельный модуль написанный, допустим, на ассемблере не составляет проблем.

Процитирую один абзац из первой части.

"В таком случае я могу предложить отменить все разновидности и модели самолетов. И оставить одну. Самую "крутую". К примеру, стратегический бомбардировщик - невидимку B - 2.Правда, людей придется перевозить в бомболюках, хотя много народу не поместится. Да и стоимость билетов учитывая то, что изготовление одного самолета стоит более двух миллиардов долларов будет немаленькая. Слишком немаленькая, чтобы практически использовать этот вид транспорта."

А как удобно управлять таким самолетом ! Ведь средства управления он должен включить в себя от всех типов самолетов. Вот и придется пилоту постоянно переключаться с панели управления вооружением на панель регулировки температуры и влажности в салоне самолета.

Важно отметить, что в этом заключается одно из ключевых различий с Виртовской линией Pascal/Modula/Oberon. На Pascal'е можно очень хорошо использовать процедурную декомпозицию, но использовать линейное программирование неудобно.

На Обероне аналоги (а тем более на Java) очень трудно писать, не используя ООП. Не ОО, имеющие ОО аналоги средства, изъяты из языка.

К примеру, без вариантной записи, используя возможность наследования расширяемых записей, т.е. классов, можно спокойно обходиться.



Тем не менее, Оберон остается универсальным языком. Конечно, не таким универсальным, как C++(в смысле не мультипарадигменный). Но природа программ, да и программистов такова, что универсальных программ, да и программистов не существует. Представьте себе игру, которая занимается не только подсчетом денег заработанных игроками, но и чтением/записью сохраненных игр посредствам работы с портами ввода - вывода(т.е. то, чем занимается низкоуровневый драйвер на ассемблере).

Или разработчика, который поочередно разрабатывает то этот драйвер, то бухгалтерский АРМ. Конечно, обе программы можно при желании написать на C++, но используемые при разработке средства будут разные. Да и программы будут уступать аналогичным, написанным на ассемблере(работать будет быстрее) и, к примеру, Delphi (разработка займет меньше времени).

С другой стороны, именно всеобъемлемость средств C++ и отсутствие четкой идеологии языка послужили его распространению. Каждый может выбирать стиль программирования по душе. C++ это сундук в котором каждый находит то сто нужно ему. В этом состоит одно из ключевых отличий от таких языков, как Pascal и Java, которые в значительной степени диктуют стиль программирования разработчику.

С одной стороны, подобное навязывание стиля - это плохо, т.к. ограничивает свободу разработчика. Но если посмотреть с другой точки зрения: вряд ли рядовой разработчик придумает что - нибудь лучшее, чем предлагают разработчики языка (среды, CASE средства и т.д.), а если и придумает, то лучше воплотить эту технологию в чем - то доступном другим разработчикам, например, ЯП.

Если ориентироваться не на разработчиков, а на задачи, то мы увидим еще одно "НО". Разные технологии в разной степени применимы к разным задачам. К примеру, ООП не дает заметного выигрыша в вычислительных математических задачах. Но ведь и заметного проигрыша тоже нет ?

К тому же каждый язык имеет свою сферу применения. Никто не предлагает писать на Perl модули ОС. Да и программисты, пишущие на Perl, и программисты, пишущие модули ОС - обычно разные люди.

В случае языка АДА подобная всеобъемлемость имела практическую необходимость. Ведь АДА была призвана заменить (и заменила) все языки (а их было несколько сотен, используемые в US DOD(министерстве обороны США) и всех организациях, работающих на него. Соответственно, такая же универсальность нужна C++ лишь в том случае, если он призван вытеснить не меньше языков. Правда, US DOD нужен был единый стандартный язык, а кому нужен C++ ?



У Страуструпа есть такая идея: "Пользователь не обязан знать ничего, кроме того подмножества языка, которое он явно применяет для написания программы"

Идея хорошая, но, к сожалению, стопроцентно нереализуемая. Элементарные операции со строками требуют знакомства с STL и шаблонами. Конечно, можно писать в стиле C, но тогда исчезают преимущества C++ перед ним. Для использования массивов с проверкой на выход индекса из диапазона необходимо использовать STL и перегрузку операций шаблона.

В одном из интервью Страуструп приводит следующие потенциальные направления развития C++:

"Параллелизм: я сторонник библиотечной реализации потоков и параллельного выполнения операций без разделения памяти.

Отображение типов: неплохо было бы обеспечить библиотечную организацию интерфейса с информацией расширенных типов.

Типизация: хотелось бы, чтобы в библиотеку Standard Library были включены функции поддержки расширенных типов, однако конкретных предложений у меня нет.

Хеш - таблицы: конечно, необходимо интегрировать некоторые варианты популярной схемы hash\_map.

Ограничения для аргументов - шаблонов: все это просто, универсально и элегантно реализуется в рамках существующего стандарта Си++.

Операторы контроля: многие из наиболее важных операторов контроля - верификация кода и обработка ошибок - можно было бы реализовать в виде шаблонов. Некоторые из них следует включить в библиотеку Standard Library.

Сопоставление с регулярными выражениями: хотелось бы видеть в стандартном варианте языка библиотеку определения соответствия шаблонам.

Сборка мусора: в стандарте Си++ нужно явно определить технологию, позволяющую игнорировать "скрытые указатели", а также конкретизировать порядок обработки деструкторов.

Графический интерфейс пользователя: хорошо было бы иметь стандартные конструкции GUI, но не знаю, насколько это реально в сложившейся ситуации.

Независимость от платформы: хотелось бы, чтобы Standard Library поддерживала более широкий набор интерфейсов с общими системными ресурсами, например, с каталогами и сокетами."

Еще одна проблема заслуживающая упоминания - это путь распространения C++. Очевидно, что тут повлияли распространенность Си, широта средств языка и областей его применения. Достаточно большой вопрос - насколько на распространение C++ повлияла рекламная поддержка ?

Вот что говорит сам Страуструп по этому поводу:

"Си++ приобрел массу приверженцев там, где это было нужнее всего: миллионы программистов используют его в качестве своего основного инструмента.

Удивительно, но этого удалось добиться, не имея <административного ресурса> и по существу без финансовых затрат. Однако, с другой стороны, сообщество сторонников Си++ оказалось разобщенным и крайне уязвимым для выпадов враждебной пропаганды. Мне кажется, все дело здесь в том, что хороший код часто остается незамеченным - даже его пользователями. Посмотрите на программы, написанные на Си++, например, на приложения Netscape или на Internet Explorer. Компании, разрабатывающие программы для решения реальных задач - в частности, для управления телекоммуникациями, контроля за различными механизмами и имитационного моделирования, - предпочитают не говорить, на каком языке написаны их приложения.

О том, что приложение написано не на C++, как правило, тоже никто не говорит. О количестве ошибок в упомянутых продуктов я не говорю.



К сожалению, это развязывает руки тем, кто пропагандирует альтернативные средства, а также любителям теоретических рассуждений.

Язык Си++ никогда не поддерживался каким - то одним лидером отрасли. Каждый крупный производитель дополняет (и так было всегда) стандарт Си++ своими собственными элементами. Си++ никогда не являлся предметом маркетинговой стратегии. Если какие - то маркетинговые мероприятия и проводились, то только организациями, продающими что - то другое (например, среду для разработки программного обеспечения), включающее в себя Си++ в качестве составляющего компонента. Сообщество Си++ скорее страдало от чрезмерной популярности этого языка: он постоянно становился <объектом нападок>, ведь в современном мире, живущем по законам коммерции, честная борьба - большая редкость.

У сообщества Си++ никогда не было координирующего центра, финансовые возможности которого позволяли бы ему заниматься популяризацией языка. Кто и из каких соображений готов голосовать за Си++? И как довести эту информацию до программистов, преподавателей, менеджеров? Буквально на днях я слушал выступление профессора перед студентами, в котором он категорически отрицал существование стандарта Си++! Жаль, что даже спустя два года после ратификации этого стандарта сплошь и рядом возникают подобные недоразумения."

Приведу и другую точку зрения.

Многие приверженцы C++ считают его языком для профессионалов. Это так.

Но они переходят дальше и производят ошибочное следствие, что единственный язык для профессионалов - это C++.

Яблоко - это фрукт. Но это не значит, что любой фрукт есть яблоко.

Поклонники Си даже сочинили стишок

"У любого ты спроси

Будь хоть трижды ламер

программируешь на Си

Ты - крутой программер"

Я cчитаю что определять крутость программиста по языку - по крайней мере некорректно. Можно писать хорошие программы на Basic и ужасные на C++. Кроме того, аргумент против C++ в религиозных войнах о его рекламной распространенности тоже появился, вероятно не с неба.

В заключение отмечу что:

В тоже время нужно признать, что C++ является наиболее мощным, вероятно за исключением Ad'ы языком из широко используемых.

Он же является на момент написания обзора наиболее популярным языком. Хотя это не бесспорно. Почему см. страницу XXX. Хотя последние годы его потеснила Java.Сейчас теснит и C#.

Он же является наиболее критикуемым языком.

Он же содержит огромное наибольшее количество недостатков и подводных камней нередко делающих программы на нем менее надежными.

P.S.

Хотелось бы знать мнения читателей по поводу:

доля того или иного языка в программной индустрии (измеренная в стоимости проектов, стоимости продаваемых программных продуктов, количестве программистов, может еще в чем...)

влияние языка на стоимость проекта, стоимость сопровождения, надежность.

принципы, по которым в индустрии выбираются языки

сферы, где существует конкуренция языков - и области, где "все поделено"

тенденции и прогнозы (например - "через N лет мы все будем в сфере A - писать на B, в сфере C - на D, и т.п."

Очень приветствуются аргументированные ответы ссылки на научные исследования,книги и статьи.

Очень интересны результаты разработок(и практический опыт приобретенный при разработке оных) однотипного(а лучше одинакового ПО) на разных ЯП и его влиянии на их стоимость/время разработки/устойчивость и скорость работы приложений/...

При этом хорошо бы рассматривать ЯП не как вещь в себе, а в связи с другим ПО(от отладчика уровня ядра до Case средств).

Интересен сравнительный анализ ЯП и компиляторов и вообще всё что с этим связанно. В частности связь с психологией. Очень интересно отличие "у нас" и " за бугром".

Интересуют также координаты людей и организаций, в чьи профессиональные и научные интересы входят ЯП.