C\C++

19 августа 2014 г.

1 Введение. Основные понятие языка

Язык программирование **c\c++** получил свое развитие с начала 1980 годов. Более рание версии назывались язком **c**. Данный язык **c\c++** удобен и использовался для написания опреационных систем. Из-за этого его называют языком системного программирования. Unix и ее програмное обеспечение так-же написанно на **c\c++**. **c++** содержит язык **c** как подмножество. Между **c** и **c++** иметеется сходство на уровне типов данных, операций, операторов, объектов, адресов и т.п. За некоторыми исключениями, а именно выделения динамической памяти: **new and delete. c++** использует теже самые последовательности вызова и возврата из функций что и в **c.** Одним из превоночальных предназначений было применение его вместо программирования на **assembler'e** — то есть облегчить жизнь программисту. Когда разрабатывали c++ были приняты все меры для сохранения этой возможности (поддерживать ассемблер). Язык c++ — это язык c, расширенный введением функций **inline**, введением классов, прегруженых операций, перегруженных имен функций, константных типов, ссылок, операций управления свободной памятью, проверки параметров функции.

1.1 Различия между c и c++

1. Типы параметров функции могут быть заданы и будут проверяться, могут выполняться преобразования типов.
2. Для выраженийс числами с плавующей точкой может использоваться, плавующаяя арифметика одинарной точности
3. Имена функции могут быть перегруженны
4. Операции могут быть перегруженны
5. Могут осуществляться **inline** подстановка функции
6. Объекты данных могут быть константами
7. Могут быть описаны объекты ссылочного типа
8. Операция **new and delete** обеспечивают свободное хранение данных в памяти
9. Класс может обеспечивать сокрытие данных, гарантированную инициализацию, определяемые пользователем преоброзования и динамическое задание типов через использование виртуальных функций
10. Имя класса являеться именем типа
11. Любой указатель, может присваиваться указателю **void*** без приведения типов

2 Алфавит языка

Не весь алфавит

= присваивание

== равно

!= неравно

Управляющие последовательности (escape последовательности) то есть специальные символьные комбинации, используемые в функциях ввода и вывода информаци. Начинаються с “\” и потом группа латинских букв:

Таблица 1: Escape последовательности

Послед-сть	Описание	Код
\a	Звонок.	007
\b	Возврат на шаг.	008
\t	Горизонтальная табуляция.	
\n	Переход на новую строку.	
\v	Вертикальная табуляция.	
\r	Возврат каретки.	
\f	Перевод формата.	
\\	Обратная дробная черта.	
\ddd	Символ набора кодов в восьмеричном представлении.	
\xddd	Символ набора кодов в шеснадцатиричном представлении.	

Когда компилятор обрабатывает программу он разбивает ее на группы символов называемых лексемы. Лексема — это еденица текста программы, которая имеет определенный смысл для компилятора, которая не может быть разбита в дальнейшем. Существует 6 классов лексем.

1. Индетификаторы и имена

   Регистр имеет значение.

2. Ключевые слова

   Это индетификаторы имеющиее определенное смысловое значение, которое не может быть использовано в качестве переменнной и констант.

3. Константы

   Константы представляют собой данные используемые только для чтения, не меняют своего значения в процессе выполнения прогрммы. Константы бывают:

   - Типизированные, с указанием типа **const int a=1;**
   - Нетипизированные, без типа #**define a 1;**

   Можно определить с помощью #**define and const.** #**define** это макрас подстановки. В качестве значения может быть использоваться простое число указанного типа или же простое вырожение, допустипое для данного типа. В зависимости от типа данных константы бывают:

   - Целые
   - Вещественный
   - Символьные
   - Строковые
   - Перечислимые

4. Строки

5. Операторы

6. Прочие разделители

Границы лексем определяються пробельными символами, а так-же другими лексемами. Количество пробелов не имеет значения. Возможно 3 способа комментирования кода:

1. //комены в одну строку
2. /* комнты не в одну строчку*/

Переменные. Они могут изменять свое значение в ходе работы програмы. Любая переменная должна иметь имя и должна быть описованна.
int a=1;
double penetretion=3.14;
string str;

Категории типов

1. **void** — нет типа
2. **function**
3. **scalar** — арифметический, перечеслимый, указатель и ссылки
4. **aggregate** — массив, струкутра, объеденения и классы

Типы данных картинка

Операнды и выражения Операнды-константы, литеры, выражение или вызов функции или более сложные выражения содержашие различные операции и скобки. Л
Каждый операнд, который представляет то или иное выражение может сам быть выражением.

1. Бывает унарное выражение, которое состоит из операнда и предшествушего ему знаку унарной операции. Формат: <знак унарной операции><выражение>
2. Бинарное выражение, состоящие из двух операндов разделенный знаком бинарной операции. Формат:<Выражение1><знак><Выражение2>.
3. Тернарное выражение,которое состоит из трех операндов разделенных знаками тернарной операции, либо вопрос ?, либо :. Формат:<Выражение1>?<Операнд 2>:<Операнд 3>

Выражение так-же делятся на 3 типа, как представленно выше. Унарные операции выполняються слева на право.

1. **Унарные операции**
   - Унарный “+”
   - Отрицание “-”
   - Inc “++”
   - Dec “–”
   - Поразрядное отрицание или дополнение“~”
   - Лоигческое отрицание “!”
   - Разодрисация “*”
   - Вычесление адреса “&”
   - Подсчет размера “sizeof”
2. **Бинарные операции**
   - “*” умножение
   - “/” деление
   - “%” остаток от деления

- “+” сложение
- “-” вычитание
- “<<” сдвиг влево
- “>>” сдвиг вправо
- “<” меньше
- “<=” меньше либо равно
- “>=” больше либо равно
- “==” равно
- “!=” неравно
- “&” поразрядное и
- “|” поразрядное или
- “^” поразрядное исключающие или
- “&&” логическое и
- “||” логическое или
- “,” последовательное вычесление
- “=” присваивание
- “*=” умножение присваивание
- “/=” деление присваивание
- “%=” no comment
- “-=” no comment
- “+=” no comment
- “<<=” no comment
- “>>=” no comment
- “&=” порозрядное и с присваиванием
- “|=” поразрядное или с присваиванием
- “^=” исключающиеся или

При вычеслении выражений тип каждого операнда может быть преобразован к другому типу. Сами преобразования могут быть не явными - это когда выполение операций и вызовы функций. Или же явными, когда выполняються операции приведения.

Приорететы операции и порядок вычесления. С высшего приоретета

1. () [] . ->
2. - + ~ ! * & ++ – sizeof - унарные
3. * / %
4. + -
5. << >>
6. < > <= >=
7. == !=
8. & (Порязрадное и)
9. ^ (Порозрядное исключающие или)
10. | (Порозрядное или)

11. &&
12. ||
13. ?: (Условная операция)
14. = *= /= %= += -= и т.п
15. , (последовательное вычесление)

Арифметические преобразование при вычеслении выражения При выполнения операции происходит автоматические преобразование типов, чтобы привести операнды выражений к общему типу или чтоб расширить, короткие велечины до размера целых велечин, используемых в машинных командах. Выполнение преобразование зависит от специфики операции и от типа операнда или операндов. Рассмотрим общие арфметические преобразования.

1. Операнды типа float преобразуються к типу double
2. Если один операнд Longdouble то второй преобразуеться к этому же типу.
3. Если один опернад double то второй так-же преобразуються к типу double
4. Любые операнды типа char and short преобразються к типу int
5. Любые операнды unsigned char and unsigned short преобразуються к unsigned int
6. Если один операнд типа unsigned long то и второй тоже будет таким
7. Если один операнд типа Long то и второй тоже к нему преобразуется
8. Если один операнд unsigned int то и второй тоже будет таким

Таким образом при вычеслении выражений, операнды преобразуються в типу того операнда, который имеет наибольший размер. Выполнение операторов присваивание, правило преобразования для следуюшего кода будут...

```
double ft,sd;
unsigned char ch;
unsigned long in;
int i;
sd=ft*(i+ch/in);
```

Операнд ch преобразуеться к unsigned int потом к unsigned long, далее по правилу i преобразуеться к unsigned long и результат операции в скобках так-же будет иметь тип unsigned long. Затем он преобразуеться к типу double. И весь результат будет иметь тип double. При преоборазование символьных переменных в целые возникает один момент, компилятор неуказывает сам должны ли переменным типам char соотвествовать числовые значение со знаком или без. Ответ на этот вопрос меняется из поколения к поколению ЭВМ содержится в их архитектуре. К примеру: крайне левый бит некой переменной char содержаший 1 перобразуеться в отрицательное целое, если 0 то положителное. На других ЭВМ такое преоразование сопровождаеться добовлением 0 с левого края. В результате чего всегда получется положительное число. В языке c++ любой символ из стандратного набора никогда не даст отрицательного. Эти символы можно спокойно использовать как положительные велечины.

Мультипликативные и аддитивные операции

Часть I
ТУТ ПИЗДЕЦ !!! ОН НЕСЕТ БРЕД

В c++ подерживаеться следующий стандартный набор мультипликатиные операций: * / % + -. Так-же унарный - или +, чтобы менять знак переменной

*-ассоциативна

Аддитивные операции + и -, групируються слева на право, и выполняються обычные арифметические преобразования. Переполнение не ообрабатываеться, тоесть информация теряется, если результат аддитивной операции не может быть представлен типом операндов после преобразования. При этом ошибки небудет.

Пример:

```
int i=30000;
int j=300000
int k;
k=i+j;
k=-чеготам.
```

Если адресс * числом, то получеться что целоые число получеться как умножение числа на размер того блока куда показывает адресс. Когда преобразованная целая велечина складываеться с велечиной указателя, то результатом являеться указатель, адресующей ячейку памяти расположенну на целую велечину дальше от исходного адреса. И новое значение указателя адресует тотже самый тип данных, что и старый указатель.

++ и – - это унарные операции присвивания они увеличивают или уменшают значения операнда на еденицу. Операнд может быть как целый так и float, а так-же указатель. В с++ допускаться два способа записи префексная и постфиксная. Если знак операций стоит перед опернадом - префиксная. Если знак оперции стоит после опернда - постфиксная.

3 Конструкция программы

1. Директивы препроцессора.

   Определяют действия перед компиляцией, а так-же включаеют инструкции, которым следует компилятор во время компиляции

2. Объявление

   Написание переменных функций, структур, классов и типов данных

3. Определения

   Тела выполняемых функций проекта

Объявление переменных:
[класс памяти][знаковость][длина]тип список-переменных

1. Класса памяти

   - auto — перменная определена в том блоке в котором она описана, и во вложенных блоках. Определенная вне всех блоков переменная видема до конца файла. И как правило слово это не пишеться
   - static — переменная существует глобально, не зависимо от того на каком блоке уровней определена. Воздействие ее до конца файла, в котором она определена
   - extern — переменая или функция определена в другом файле, а объявление представляет собой ссылку действуюшую в рамках одного проекта.
   - register — используеться только для типов char and int. Означает, что переменнная при возможности храниться в регистре процессора.

   1 и 4 классы не допу каються к явному указанию на внешному уровне.

2. Знаковость

   Может быть указана только для перечеслимых типов или порядковых типов: char and int.

   - signed

- unsigned

3. Длина

   Определена для Int, float, double

   - Отсутсвует(по умолчанию)
   - short
   - long

Дейтвие этих модификаторов зависит от компилятора и апаратной платформы. На IBM PC-short int and int совпадают и занимают 2 байта, а Long int требует 4 байта

4 Описание функции

Формат объявления:
<Тип функции> <имя >(тип1 параметр1, ... , типn параметрn);
Объявление необходиом для функций, которые описанны ниже.
Пример:

```
float f1(double t, double v)
{

    return t+v;

}
extern char fo();// функция в другом файле
f2();//прототип, а сама функция вместе с телом определно ниже
```

Для вызова функции из другого файла можно использовать:

1. Extern;
2. #include<имя файла.h> если свой класс то в ковычках

Пример:

```
int f1(int n)// если функция без типа, то по умолчанию int
{

    extern int f4(int);
    return f4(n);

}//функция определена во внешнем файла и должна быть доступна к моменту сборки приложения.
```

Объявление типа позволяет создать свой тип, оно состоит в присвоении некотрому базовому или составному типу языка c++.

```
struct list
{

    char name[20]
    long int phone;

}
struct list mylist[20]
```

Описан массив структур типа List с именем Mulist из 20 элементов.
Есть еще typedef дает другое имя уже сушествуем струтурам или типам. typedef float real.
Так-же есть #define. #define cin read
Определение функции задает ее тело, которое представляет собой состовной оператор или блок, содержащий что-то там. Определение функции задает имя тип возращаемого значения и атрибуты ее формальных параметров.

```
int f(int a, int b)
{
    return(a+b)/2;
}
```

Допускается спецификацции класса памяти, static или extern.

Часть II
Я ОТКАЗЫВАЮЬ ДАЛЬШЕ ПИСАТЬ

Содержание