**Представление данных.**

Данные это формализованное представление информации. В компьютерной программе данные – это значение объектов программы. Данные С++ могут быть в виде констант и переменных.

***Данные, которые не изменяются в программе, называются константами. Данные зафиксированные в программе и изменяемые в ней в процессе обработки являются переменными.***

Переменные, перед тем как их использовать в программе должны быть объявлены. Объявление переменной состоит в первую очередь из указания ее типа.

***Понятие типа данных.***

Тип данных характеризует:

* ***внутреннее представление данных в памяти компьютера;***
* ***набор допустимых операций (действий);***
* ***множество допустимых значений.***

***Классификация данных языка С++. Основные и производные типы***

Все типы данных можно подразделить на простые (основные) — они предопределены стандартом языка, и сложные (или составные) — задаются пользователем. Данные простого типа нельзя разложить на более простые составляющие без потери сущности данного. Простые типы данных создают основу для построения более сложных типов: массивов, структур, классов. Простые типы в языке C++ — это целые, вещественные типы, символьный и логический тип и тип **void**.

Для определения и описания переменных основных типов используются следующие ключевые слова:

* **char** (символьный);
* **short** (короткий целый);
* **int** (целый);
* **long** (длинный целый);
* **float** (вещественный);
* **double** (вещественный с удвоенной точностью);
* **void** (отсутствие значения).

В табл. 1.7 приведены основные типы данных с диапазоном значений и назначением типов для компиляторов семейства IBM PC/XT/AT.

В таблице базовыми типами являются: ***char, int, float, double, void.*** Остальные получаются из них с использованием **модификаторов** типа:

***unsigned*** (беззнаковый), ***signed*** (знаковый), ***long*** (длинный), ***shot*** (короткий).

***Целые типы***

Целый тип данных предназначен для представления в памяти компьютера обычных целых чисел. Основным и наиболее употребительным целым типом является тип ***int****.* Гораздо реже используют его разновидности: ***short*** (короткое целое) и ***long*** (длинное целое).

Таблица 1.7

ОСНОВНЫЕ ТИПЫ ДАННЫХ

|  |  |  |  |
| --- | --- | --- | --- |
| **Тип** | **Размер**  **(биты)** | Диапазонзначений | Назначение типа |
| unsigned char | 8 | 0…255 | Небольшие целые числа и коды символов. |
| char | 8 | -128…127 | Очень малые целые числа и ASCII-коды. |
| enum | 16 | -32768…32767 | Упорядоченные наборы целых значений. |
| unsigned int | 16 | 0…65535 | Большие целые и счётчики циклов. |
| shot int | 16 | -32768…32767 | Небольшие целые. Управление циклами. |
| int | 16 | -32768…32767 | Небольшие целые. Управление циклами. |
| unsigned long | 32 | 0… 232-1 | Астрономические расстояния. |
| long | 32 | -231 …231-1 | Большие числа, популяции. |
| float | 32 | 3.4E-38 …3.4E+38 | Научные расчёты (7 значащих цифр) |
| double | 64 | 1.7E-308 … 1.7E+308 | Научные расчёты (15 значащих цифр) |
| long double | 80 | 3.4E-4932 …1.7E+4932 | Финансовые расчёты (19 значащих цифр) |
| void |  |  |  |

По-умолчанию все целые типы являются ***знаковыми***, то есть старший бит в таких числах определяет знак числа: 0 — число положительное, 1 — число отрицательное. Для представления отрицательного числа используется дополнительный код.

Кроме знаковых чисел на C++ можно использовать *беззнаковые*. В этом случае все разряды участвуют в формировании целого числа. При описании беззнаковыхцелыхпеременных добавляется слово ***unsigned*** (без знака).

Для 32-разрядных компиляторов, которые не делают различия между целыми типами ***int* и *long*** целые типы представлены для сравнения в табл.1.8.

Таблица 1.8

*Целые типы для 32-разрядных компиляторов*

|  |  |  |
| --- | --- | --- |
| ***Тип данных*** | ***Размер, байт*** | ***Диапазон значений*** |
| **char** | **1** | **-128 ... 127 (-27 – 27-1)** |
| **short** | **2** | **-32768 ... 32767 (-215 – 215-1)** |
| **int** | **4** | **-2147483648 ... 2147483647 (-231 – 231-1)** |
| **long** | **4** | **-2147483648 ... 2147483647 (-231 – 231-1)** |
| **unsigned char** | **1** | **0 ... 255 ( 0 – 28-1)** |
| **unsigned short** | **2** | **0 ... 65535 ( 0 – 216-1)** |
| **unsigned int** | **4** | **0 ... 4294967295 ( 0 – 232-1)** |
| **unsigned long** | **4** | **0 ... 4294967295 ( 0 – 232-1)** |

***Символьный тип***

В стандарте C++ для представления символьной информации есть два типа данных, пригодных для этой цели, — это типы ***char*** и ***wchar\_t*.**

Тип ***char*** используется для представления символов в соответствии с системой кодировки *ASCII* (*American Standard Code for Information Interchange — Американский стандартный код обмена информации*). Это семибитовый код, его достаточно для кодировки 128 различных символов с кодами от 0 до 127. Символы с кодами от 128 до 255 используются для кодирования национальных алфавитов, символов псевдографики и др.

Тип ***wchar\_t*** предназначен для работы с набором символов, для кодировки которых недостаточно 1 байта, например, Unicode. Размер типа ***wchar\_t*** обычно равен 2 байтам. Если в программе необходимо использовать строковые константы типа ***wchar\_t***, то их записывают с префиксом ***L***, например, ***L"Слово".***

Таким образом, значениями символьных данных являются целые числа – значения их внутреннего кода. Однако в операторах ввода/вывода фигурируют сами символы, что иллюстрирует следующая программа:

***#include <iostream.h>***

***char c,b ;***

***void main()***

***{ c=’\*’; b= 55;***

***cout<<c<<’\t’<<b<<’\t’;***

***cout<< (int)c;}***

Результатом будет:

***\* 7 42***

Вывелись символы ***'\*'*** и символ с кодом 55 – символ '7'. И код символа ***'\*'***, как результат приведения типа символа к целому типу.

***Логический тип***

Логический (булевый) тип обозначается словом ***bool***. Данные булевого типа могут принимать только два значения: ***true*** и ***false*** и занимают в памяти 1 байт. Значение **false** обычно равно числу 0, значение ***true*** — числу 1.

***Вещественные типы***

Особенностью вещественных (действительных) чисел является то, что в памяти компьютера они практически всегда хранятся приближенно.

Имеется три вещественных типа данных: ***float, double*** и ***long double***. Основным считается тип ***double.*** Так, все математические функции по умолчанию работают именно с типом ***double***. В табл. 1.9 приведены основные характеристики вещественных типов [2].

Таблица 1.9

*Основные характеристики вещественных типов*

|  |  |  |  |
| --- | --- | --- | --- |
| ***Тип данных*** | ***Размер, байт*** | ***Диапазон***  ***абсолютных величин*** | ***Точность, количество десятичных цифр*** |
| float | 4 | от 3.4Е—38 до 3.4Е+38 | 7 — 8 |
| double | 8 | от 1.7Е—308 до 1.7Е+308 | 15 — 16 |
| long double | 10 (12) | от 3.4E-4932 до 1.1E+4932 | 19 — 20 |

Рекомендуется везде использовать тип ***double***. Работа с ним всегда ведётся быстрее, меньше вероятность заметной потери точности при большом количестве вычислений.

***Тип void***

Тип ***void*** — самый необычный тип данных языка C++. Нельзя определить переменную этого типа. Тем не менее, это очень полезный тип данных. Он используется:

* ***для определения функций, которые не возвращают результата своей работы;***
* ***для указания того, что список параметров функции пуст;***
* ***а так же этот тип является базовым для работы с указателями***.

Указатель на тип ***void - у***казатель ни на что, но который может быть приведен к любому типу указателей. Так всё программирование с использованием Win32 API построено на применении указателей на тип **void**.

***Новое обозначение типа***

Используя спецификатор ***typedef*** можно в программе вводить удобное обозначение для сложных обозначений типов по следующему правилу:

***typedef имя типа новое имя типа.***

В следующем примере:

***typedef unsigned char cod; cod symb;***

объявлена переменная ***symb*** типа ***unsigned char.***

С помощью операций \*, &, [ ], ( ) и механизмов определения структурированных типов можно создавать производные типы.

***Представление некоторых форматов производных типов:***

1. ***type имя [ ]*** – массив элементов типа type, например: ***long m [5];***
2. ***type1 имя (type2)*** - функция с аргументом ***type2***  и результатом типа ***type1***:

3. ***type \* имя*** - указатель на объект типа ***type***, например: ***char\*ptr***;

4. ***type\* имя [ ]*** –массив указателей на объекты типа ***type***: ***int\* arr[10];***

5. ***type (\*имя) [ ]*** –указатель на массив объектов типа ***type***: ***int (\*ptr) [10];***

6. ***type1\* имя (type2)*** – функция, с аргументом типа ***type2*** и возвращающая указатель на объект типа ***type1***;

7. ***type1 (\*имя) (type2)*** - указатель на функцию с аргументом типа type2, возвращающую значение типа ***type1***;

1. ***type1\* (\*имя) (type2)*** – указатель на функцию с аргументом ***type2***, возвращающую указатель на объект типа ***type1***;
2. ***type & имя = имя\_объекта типа\_type*** – определение ссылки;
3. ***type (&имя) (type2)*** – ссылка на функцию с аргументом ***type2***, возвращающую результат типа ***type1***;

11. ***struct имя{ type1 имя1; type2 имя2; };-*** объявление структуры;

1. ***union имя {type1 имя1; type2 имя2;};*** - объявление объединения;
2. ***class имя { type1 имя1; type2 имя2 ( type3);};*** - определение класса;

Таким образом, типы данных можно разделить на ***скаляры, агрегаты и функции.***

***Скаляры* –** это арифметические типы, перечисляемые типы, указатели и ссылки. ***Агрегаты*** (структурированные типы) – массивы, структуры, объединения и классы. ***Функции*** подробно рассмотрим в разделе 3.

*Объявление переменных и констант в программе*

Форма объявления переменных заданного типа:

*имя\_типа список имен переменных;*

Например: float x, X, cc2, pot\_b;

Переменные можно инициализировать, то есть задавать им начальные значения при их определении. Форма определения начальных значений проиллюстрирована на примере:

*unsigned int year = 1999;*

*unsigned int year ( 1999 );*

Последнее определение с инициализацией разрешено только в функциях.

Допустимы следующие формы объявления именованных констант:

1. константы в объявлении перечисляемого типа;
2. с помощью спецификатора *const* :

*const имя\_типа имя\_константы = значение*

пример: *const long M = 99999999*;

1. определение константы в препроцессорной директиве *define*

пример: *#define имя\_константы значение\_константы*

**Объекты программы и их атрибуты.**

***Переменная* - э**то именованная область памяти. Имя переменной – это ссылка на некоторую область памяти. Переменная - частный случай *леводопустимого выражения****.***

***Понятия леводопустимых и праводопустимых выражений***.

*Леводопустимое выражение* – это конструкция для обращения к некоторому участку памяти, куда можно поместить значение (***lvalue, l-значение*).**

Понятие *леводопустимого выражения* включает всевозможные формы обращения к некоторому участку памяти с целью изменения его содержимого. Название сформировалось по причине, что *леводопустимые выражения* располагаются слева в операторе присваивания.

Примеры *леводопустимых выражений*:

1. имена скалярных переменных;
2. имена элементов массивов;
3. имена указателей;
4. ссылки на ***lvalue*** (синонимы ***lvalue***);
5. имена элементов структурированных данных:

***имя структуры . имя элемента;***

***указатель на структуру -> имя элемента элемента;***

6) выражения с операцией '**\***' - разыменования указателя:

***int i, \*p =&i; \*p=7;***

здесь объявляется переменная и указатель на эту переменную, а затем с помощью операции разыменования указателя осуществляется доступ к ячейке памяти переменной.

7) вызовы функций возвращающих ссылки на объекты программы.

Выражения, которые могут располагаться в правой части оператора присваивания называются *праводопустимыми* выражениями.

Примеры *праводопустимых выражений*:

1. любое арифметическое, логическое выражение;
2. имя константы;
3. имя функции (указатель константа);
4. имя массива (указатель константа);
5. вызов функции, не возвращающей ссылки;

В дальнейшем рассмотрении, в качестве ***lvalue*** будет рассматриваться переменная как объект программы.

Кроме типов для переменных явно или по умолчанию определяются:

* класс памяти (задает размещение объекта);
* область действия, связанного с объектом идентификатора (имени);
* видимость объекта;
* продолжительность существования объекта;
* тип компоновки (связывания).

Все перечисленные атрибуты (свойства) взаимосвязаны и должны быть либо явно указаны, либо они выбираются по контексту неявно при определении переменной. Рассмотрим их подробнее.

***Тип***как уже указывалось выше, определяет размер памяти, выделяемого для значения объекта, правила интерпретации двоичных кодов значений объектов.

***Класс памяти*** определяет размещение объекта в памяти и продолжительность его существования**.**

Явно задать ***класс памяти*** можно с помощью спецификаторов: ***auto, register, static, extern.***

Ниже указаны спецификаторы ***класса памяти*** и соответствующее им место размещения объекта:

* ***auto* -** автоматически выделяемая, локальная оперативная память - ***сегмент стека*** (временная память). Спецификатор ***auto*** может быть задан только при определении переменной блока, например в теле функции. Локальная продолжительность существования. Этим объектам память выделяется при входе в блок и освобождается при выходе из него. Вне блока переменные класса ***auto*** не существуют.
* ***register* -** автоматически выделяемая по возможности регистровая память (***регистры процессора***). Спецификатор ***register*** аналогичен **auto,** но для размещения переменной используется не оперативная память, а регистры процессора. Если регистры заняты другими переменными, переменные класса ***register*** обрабатываются как объекты класса ***auto***.
* ***static*** - статическая продолжительность существования, память выделяется в ***сегменте данных,*** объект существует до конца программы. Внутренний тип компоновки – объект будет существовать в пределах того файла с исходным текстом программы, где он определен. Этот класс памяти может быть приписан как переменным, так и функциям.
* ***extern*** - внешняя, глобальная память, выделяется в ***сегменте данных***. Объект класса ***extern*** обладает статической продолжительностью существования и внешним типом компоновки. Он - глобален, то есть, доступен во всех файлах программы. Этот класс может быть приписан как переменным, так и функциям.

*Класс памяти*,и соответственноразмещение переменной (в стеке, в регистре, в динамически распределенной памяти, в сегменте данных) зависит как от синтаксиса определения, так и от размещения определения в программе.

***Область действия (ОД) идентификатора (имени)*** *-* это часть программы, в которой можно обращаться к данному имени (сфера действия имени в программе).

Рассмотрим все случаи:

1. имя определено в блоке (локальные переменные)**: ОД** - от точки определения до конца блока;
2. формальные параметры в определении функции (формальные параметры – это те же локальные переменные функции): **ОД** параметров– блок тела функции;
3. метки операторов: **ОД** меток– блок тела функции;
4. глобальные объекты: **ОД** глобальныхобъектов- вся программа от точки их определения или от точки их описания;
5. формальные параметры в прототипе функции: **ОД** параметров - прототип функции.

***Область видимости (ОВ)***

Понятие области видимости понадобилось в связи с возможностью повторных определений идентификатора внутри вложенных блоков или функций. В этом случае разрывается связь имени с переменной, то есть с участком памяти данной переменной, она становится "невидимой" внутри вложенного блока, хотя сфера действия имени сохраняется.

***ОВ*** – это часть программы, в которой обращение к имени переменной позволяет обратиться к участку памяти, связанному с данной переменной.

***ОВ*** может быть только меньшей или равной ОД. Следующая программа проиллюстрирует данную ситуацию [3]**:**

***#include<iostream.h>***

***int k=0;***

***void main()***

***{int k=1;***

***{cout << k;***

***char k = 'A'; cout << k;***

***cout << ::k; cout << k ;}***

***cout << k;}***

Результат выполнения программы: **1A0A1**

Объявлена глобальная переменная ***int k***. Затем в главной функции определяется локальная переменная с тем же именем с именем и того же типа. Глобальная переменная становится "невидимой" в теле функции. Обращаться к "невидимой" внутри функции переменной можно используя операцию доступак внешнему объекту ***::k****.*

В главной функции определен внутренний блок, в котором определена локальная переменная внутреннего блока с тем же именем ***char k*.** От точки определения символьной переменной до конца внутреннего блока локальная целочисленная переменная ***int k*** становится недоступной. После выхода из блока видимость (доступность) данной переменной восстанавливается.

***Продолжительность существования***

Продолжительность существования **-** это период, в течение которого идентификаторам в программе соответствуют конкретные объекты в памяти.

Определены три вида продолжительности: ***статическая, локальная и динамическая.***

Объектам со ***статической продолжительностью существования*** память выделяется в начале выполнения программы и сохраняется до конца программы.

Статическую продолжительность имеют все функции и файлы.

Все глобальные переменные (т.е. объявленные вне всех функций) обладают статической продолжительностью существования.

Локальные переменные (объявленные в функциях), со спецификатором ***statiс*** также имеютстатическую продолжительность**.** Статическая переменная, локализованная в функции, не теряет своего значения при отработке функции. Инициализируется статическая переменная только при первом вызове функции, при втором вызове и последующих вызовах – значением этой переменной будет то, что сохранилось в памяти при предыдущем вызове.

Глобальные и статические переменные по умолчанию инициализируются нулями.

Если в функции имеется описание переменной со спецификатором ***extern****,* это означает, что эта переменная глобальная и ее определение дано в другом месте вне функции. Такая переменная имеет статическую продолжительность существования.

***Локальной продолжительностью существования*** обладают автоматические (локальные) переменные, объявленные в блоке. Такие переменные создаются при каждом входе в блок, где они определены и уничтожаются при выходе. Локальные переменные должны инициализироваться только явно, иначе их начальные значения не предсказуемы. Область действия локального объекта – блок. Спецификатор класса ***auto*** всегда избыточен, так как этот класс по умолчанию приписывается всем объектам, определенным в блоке.

Объекты с ***динамической продолжительность*** ***существования*** создаются и уничтожаются с помощью операторов в процессе выполнения программы по желанию программиста. Память таким переменным выделяется в области динамически распределяемой памяти, называемой **кучей.**

Для создания объекта используются операция  ***new*** или функции *malloc()* и *сalloc(),* а для уничтожения - операция ***delete*** или функция *free ().*

Операция: ***new имя\_типа*** или ***new имя\_типа инициализатор***

выделяет и делает доступным участок памяти для объекта данного типа и в выделенный участок заносит значение инициализатора, что не обязательно. Операция возвращает адрес первого байта выделенного участка, или нулевое значение в случае неудачи. Если надо определить динамическую переменную типа ***int*,** следует объявить указатель на ***int*** и ему присвоить результат операции **new,** например: ***int\*r = new (15);***

В дальнейшем доступ к выделенному участку памяти обеспечивается выражением ***\*r.***

Продолжительность существования выделенного участка – от точки создания до конца программы или до явного освобождения памяти операцией ***delete r;***

***Тип компоновки***

Если программа состоит из нескольких файлов, каждому имени, используемому в нескольких файлах, может соответствовать:

1) один объект, общий для всех файлов, или

2) один и более объектов в каждом файле.

Файлы программы могут транслироваться отдельно, и в этом случае возникает проблема установления связи между одним и тем же идентификатором и единственным объектом, которому он соответствует. Таким объектам компоновщик обеспечивает ***внешнее связывание*** при объединении отдельных модулей программы (первый случай).

Для объектов, локализованных в файлах, используется ***внутреннее связывание***(второй случай)**;**

Тип компоновки компилятор устанавливает по контексту.

***Определения и описания объектов программы.***

Все описанные выше атрибуты (тип, класс памяти, ОД и так далее) приписываются объекту при его определении (объявлении) или при его описании, а также контекстом определения и описания.

В чем разница между определением или описанием.

При ***определении (definition)*** или ***объявлении*** объектаему дается имя, и устанавливаются атрибуты объекта, в соответствии с которыми выделяется нужный объем памяти и определяется формат внутреннего представления объекта. При ***определении*** происходит связывание имени объекта с участком памяти. ***Определение*** выполняет инициализацию объекта. ***Определение*** объекта может быть только одно в программе.

***Описание*** или ***декларация*** *(****declaration****)* дает знать компилятору, что объект определен и напоминает свойства объекта (в основном компилятор интересуют типы).

Обычно, ***описание*** - это представление в конкретной функции уже объявленного где – то объекта. ***Описаний***объекта может быть несколько в программе.

Нередко описание и определение по внешнему виду совпадают. Не совпадаютони в следующих случаях:

1)описание – прототип функции

2)описание содержит спецификатор ***extern***

3)описывается класс или структурный тип

4)описывается статический компонент класса

5)описывается имя типа с помощью **t*ypedef***

***Определения (объявление) переменных*** заданного типа имеет следующий формат:

***s m тип имя1 иниц.1, имя2 иниц.2, … ;***

где ***s* -** спецификатор класса памяти **auto , static , extern , register,**

***m –***модификатор ***const*** или ***volatile***:

***const* -** указывает на постоянство объекта,

***valatile* –** указывает на изменчивость объекта без непосредственного обращения к нему.

Синтаксис ***инициализации*** переменной, определяющий на этапе компиляции начальное значение переменной**:**

***имя = инициализирующее выражение*;**

либо:

***имя (инициализирующее выражение)* –** применяется только в функциях.

**Выражения и преобразования типов**

Выражение – это правило получения нового значения. Выражения формируются из операндов, операций и круглых скобок. Порядок применения операций к операндам определяется рангами операций и их ассоциативностью. Круглые скобки, как и в математических выражениях, позволяют изменить порядок выполнения операций.

При выполнении операций нужно учитывать особенности представления в программе данных разных типов, являющихся операндами выражений. В связи с этим рассмотрим преобразование и приведение типов, допустимые в С++.

***Явное приведение типа***

Синтаксис функционального приведения типа:

***type(выражение)***

Примеры: ***int(*'*\**'), *float (5/3), char(65).***

Функциональное преобразование типа не может применяться для типов, не имеющих простого имени. Например, конструкции:

***unsigned long (15/7)*** или ***char\*(0777)***

вызовут ошибку при компиляции.

Кроме функционального преобразования типа может использовать каноническую операцию приведения к требуемому типу. Для ее представление обозначение типа заключается в круглые скобки. Такая операция может применяться и для типов, имеющих сложное обозначение. Допустима записи: ***(unsigned long )15/7*** или ***(char\*) 0777***

Другую возможность явного преобразования типов со сложным обозначением дает введение собственных обозначений с помощью ***typedef***:

***typedef* *unsigned long ul; ul(15/7);***

***typedef*** ***char\* pchar*; *pchar(0777);***

В последних версиях языка С++ [5] введены еще четыре операции явного преобразования типа, имеющих следующий формат:

***название\_ cast <целевой тип> операнд***

*dynamic\_cast* - операция приведения типа с проверкой допустимости на этапе выполнения программы;

*static\_cast* – операция приведения типов с проверкой допустимости приведения во время компиляции;

*reinterpret\_cast* – операция приведения типов без проверки допустимости приведения;

*const\_cast* – операция приведения типов, которая аннулирует действие модификатора *const*.

Приведенным выше примерам соответствуют следующие выражения:

*static\_cast <unsigned long>(15/2);* и *static\_ cast <char\*>(0777);*

При преобразовании типов существуют некоторые ограничения. Но прежде чем остановиться на них, рассмотрим стандартные преобразования типов, выполняемые по умолчанию.

*Стандартные преобразования типов*

При вычислении выражений операции требуют, чтобы операнды имели соответствующий тип, а если требования не выполнены, производится неявное приведение типа.

Неявное приведение типов происходит при инициализации, когда тип инициализирующего выражения приводится к типу определяемого объекта. То же относится ко всем формам операции присваивания, происходит неявное преобразование типа выражения к типу левого объекта. Реально типы могут преобразовываться один в другой многими способами. Рассмотрит на данном этапе лишь некоторые из них. Преобразования типов, выполняемые неявно:

* преобразования в логические значения: в значения типа *bool* преобразуются обобщенные целые числа (включая и символы), вещественные числа и указатели; ненулевые значения преобразуются в *true*, а нулевые - в *false*;
* преобразование указателей:

любой указатель может быть неявно преобразован в *void\**;

значение 0 преобразуется в любой указательный тип;

неконстантный указатель преобразуется в константный указатель того же типа.

* преобразование операндов в арифметических выражениях

Последнее преобразование рассмотрим подробнее. При преобразовании нужно различать преобразования, изменяющие внутреннее представление данных и преобразования, изменяющие только интерпретацию внутреннего представления. Например, при преобразовании *unsigned int* в *int* изменяется только интерпретация внутреннего представления. При преобразовании *double* в *int* изменяется как длина участка памяти для внутреннего представления, так и способ кодировки. При таком преобразовании возможен выход за диапазон допустимых значений типа *int* , потеря значимости, точности и так далее. Именно поэтому в программах рекомендуется с осторожностью применять преобразования типов.

В арифметических выражениях происходит автоматическое преобразование типа операндов к общему типу – наиболее высокому типу согласно иерархии типов. Преобразование происходит по схеме, изображенной на рис.3:

*long double*

*doubl float*

*unsigned long int*

*long int char*

*unsigned int unsigned shot unsigned char*

Рис.3. Схема последовательных преобразований типов операндов

*Преобразования, гарантирующие сохранение значимости.*

Используя в программе арифметические выражения, следует учитывать, что некоторые из них приводят к потерям информации и изменению числового значения. На рис.4 в соответствии со стандартом языка С++ представлены стрелочками преобразования, гарантирующие сохранение точности и неизменность числового значения.

unsigned char

char

unsigned shot

float

shot

unsigned int

double

int

unsigned long

long double

long

Рис.4. Последовательности преобразований типов, гарантирующие сохранение значимости