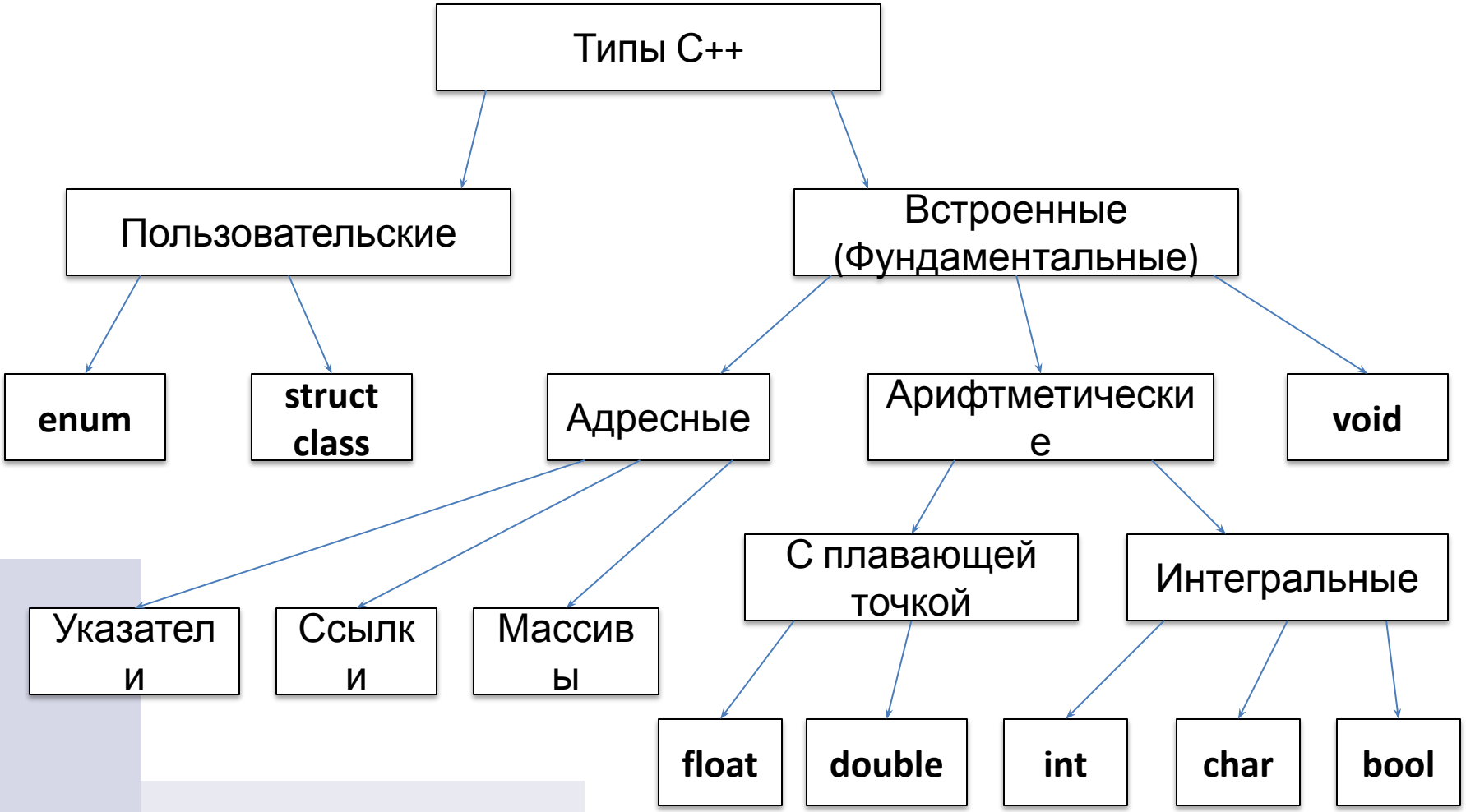
Лекция 7

Приведение типов

Классификация типов



Виды приведений

- Явное

  Преобразование произвольных типов с указанием оператора приведения

  - **C++**-style
    - static_cast
    - reinterpret_cast
    - const_cast
    - dynamic_cast
  - **C**-style

- Неявное

  Преобразование встроенных типов без явного указания оператора приведения

Явное C++-style приведение

- Статическое приведение **static_cast**

  Преобразование «родственных» типов:

  - указатель на потомка в указатель на родителя (восходящее преобразование)
  - указатель на **void** в любой другой указатель
  - дробного в интегральный
  - интегральный в перечисление

- Слепое приведение **reinterpret_cast**

  Преобразование с сохранением битового представления, невыполнимое явно или при помощи static_cast

- Снятие квалификаторов **const_cast**

  Снятие квалификаторов **const** и **volatile**

- Полиморфное приведение **dynamic_cast**

  Нисходящее преобразование при наследовании

Примеры явного C++ приведения

```
int d = static_cast < int >( 7.5 );

int const * cpd = &d ;

int * pd = const_cast < int * >( cpd );

float f = static_cast < float > ( d );

float * pf = reinterpret_cast<float*>( pd );

float * pf2 = static_cast<float*>( static_cast<void*>(pd) );

float h = *reinterpret_cast<float*>( &d );
```

Явное C-style приведение

- Cast-форма (T)e

  Любое преобразование, которое может быть выражено через комбинацию static_cast, reinterpret_cast и const_cast.

```
int n = (int)7.6 ;
void const * const pv = 0 ;
float * pr = (float *)pv;
int n_ptr = (int)pv ;
```

- Функциональная форма T(e)

  Для встроенных типов идентична (T)e

  Для пользовательских типов вызывает соответствующий конструктор

  Для встроенных типов форма T() означает инициализацию нулем

```
int n = int(7.6) ;
int q = int();
typedef float* float_ptr ;
float * pr = float_ptr(7);
```

Неявное преобразование

- Применение

  Инициализация переменных

  Преобразование аргументов функций

  Вычисление значения выражения

- Приводятся типы:

  Арифметические (без потери точности)

  Перечислимые к интегральным

  Указатели к указателям на **void**

  Произвольные типы к идентичным с квалификатором **const**

Указатель на **void**

- Переменные типа void* нельзя разыменовывать
- Указатель на любой объект можно присвоить переменной типа **void***
- Переменную типа **void*** можно присвоить другой переменной типа **void***
- Две переменных типа **void*** можно сравнивать на равенство и неравенство
- **void*** можно **явно** преобразовать в другой тип

```
int n = 0 ;
int * pn = &n ;

*pn = 7 ; // Можно

void * pv = pn ;

*pv = 9 ; // Нельзя!

float * pf = pv ;
float * pq = pn ;

float * pr = (float *)pv;
```

Перегрузка функций

- Перегрузка по аргументам

  Функции с одинаковым именем, но разными типами аргументов, являются **перегруженными**. Выбор перегруженной функции осуществляется при вызове по типу аргументов.

- Перегрузка по возвращаемому значению

  Функции не перегружаются по возвращаемому значению. Наличие двух функций с одинаковыми именами и аргументами и различающихся только возвращаемым значением, является ошибкой.

- Перегрузка и область видимости

  Перегруженными являются одноименные функции, расположенные в одной области видимости. Для функций из разных областей видимости действует правило сокрытия имен.

Правила разрешения вызова

1. Точное соответствие типа

   Тип фактического аргумента соответствует типу формального полностью или с тривиальным преобразованием (массив к указателю, T к const T)

2. Соответствие с помощью продвижения в группе

   Фактический и формальный аргумент принадлежат одной группе арифметических типов, и формальный «больше» фактического. Например, float->double, char->int, short->int и т.п.

3. Соответствие с помощью неявного преобразования

   Существует неявное преобразование, приводящее фактический аргумент к формальному

4. Соответствие с помощью пользовательского преобразования

   Существует объявленное пользователем преобразование, приводящее фактический аргумент к формальному

5. Соответствие неуказанным аргументам

   Формальный аргумент объявлен как ...

Разрешение вызова

- Разрешение вызова при одном аргументе

  Осуществляется поиск функции с соответствующим аргументом в порядке указания правил.

  Если для правила не нашлось соответствия, переходим к следующему

  Если для правила найдено одно соответствие, оно и используется

  Если для правила есть более одного соответствия, это считается неоднозначностью вызова (ambiguous call)

- Разрешение вызова при многих аргументах

  Осуществляется поиск по каждому аргументу

  Выбирается функция с наиболее хорошим совпадением одного

Примеры перегрузки

```
void print (int);
void print (const char *);
void print (double);
void print (long);
void print (char);

void f ( char c, int i, short s, float f )
{
    print (c);    // точное, (5)
    print (i);    // точное, (1)
    print (s);    // продвижение, (1)
    print (f);    // продвижение, (3)
    print ('a'); // точное (5)
    print (49L); // точное (4)
    print (0);    // точное (1)
    print ("a"); // точное (2)
}
```