**Еквівалентність типів**

Існує кілька схем для визначення того, чи є типи двох об'єктів еквівалентними. Дві схеми, найбільш часто використовувані, називаються *структурна еквівалентність типів і іменна еквівалентність типів*. У відповідності зі **схемою структурної еквівалентності типів** два об'єкти відносяться до одного і того ж типу тільки в тому випадку, якщо їх компоненти мають однакові типи. У відповідності зі **схемою іменної еквівалентності типів** два об'єкти мають один і той же тип тільки в разі їх визначення з використанням імені того ж типу.

Більшість реалізацій мови Сі використовують схему структурної еквівалентності типів. Однак в книзі (Ritche, D.M. 1980 / The C Programming Language - Reference Manual / AT & T Bell Laboratories, Murray Hill, N.J. 07974) питання про еквівалентність типів ігнорується, і за будь-якої реалізації може бути обрана своя схема визначення еквівалентності типів. **Отже, цілком можливо, що результати правильно працюючої програми стануть невірними при заміні компілятора!**

**Перетворення типів**

В операторах і виразах повинні використовуватися змінні і константи тільки одного типу. Якщо все ж ми змішуємо типи в одному вираженні, то *компілятор* мови Сі не вважає програму неправильною, як це сталося б при програмуванні на мові Паскаль. Замість цього *компілятор* використовує набір правил для автоматичного перетворення типів. Це дуже зручно, але може виявитися і небезпечним, особливо якщо ми допустили змішання типів ненавмисно. Наведемо кілька основних правил, що стосуються перетворення типів:

1. Якщо операція виконується над даними двох різних типів, обидві величини наводяться до вищого з двох типів. Цей процес називається **підвищенням типу.**
2. Послідовність імен типів, упорядкованих від вищого типу до нижчого, виглядає так: double, float, long, int, short, char. Використання ключового слова unsigned підвищує ранг відповідного типу даних зі знаком.
3. В операторі присвоєння кінцевий результат обчислення виразу в правій частині приводиться до типу змінної, якій має бути присвоєно це значення. Даний процес може привести до підвищення типу, як описано вище, або до зниження, при якому величина приводиться до типу даних, яка має нижчий пріоритет.

*Підвищення типу* зазвичай відбувається гладко, в той час як зниження може привести до ускладнень. Причина цього проста: все число цілком може не поміститися в елементі даних нижчого типу*. Змінна* типу char може мати ціле значення 101, але не 22225.

Приклад, наведений нижче, ілюструє застосування цих правил:

/\*перетворення\*/

int main( )

{

char ch;

int i;

float f1;

f1=i=ch='A'; /\*\*\*8\*\*\*/

printf("ch=%c,i=%d,f1=%2.2f\n",ch,i,f1);

ch=ch+1; /\*\*\*10\*\*\*/

i=f1=f1+2\*ch; /\*\*\*11\*\*\*/

f1=2.0\*ch+i; /\*\*\*12\*\*\*/

printf("ch=%c,i=%d,f1=%2.2f\n",ch,i,f1);

ch=2.0e30;/\*\*\*14\*\*\*/

printf("Теперь ch=%c\n",ch);

}

Виконавши програму "перетворення", отримаємо такі результати:

ch=A,i=65,f1=65.00

ch=B,i=197,f1=329.00

Тепер ch=

**Розбір програми**

Рядки 8 і 9: величина ' A ' присвоюється символьній змінній ch. Змінна i отримує ціле значення, яке являється перетворенням символу ' A ' в ціле число, тобто 65. Змінна f1 отримує значення 65.00, яке являється перетворенням числа 65 в число с плаваючою крапкою.

Рядки 10 і 13: значення символьної змінної ' A ' що перетворюється в ціле число 65, до я кого потім додається 1. Після цього отримане в результаті число 66 перетворюється в код символу В і поміщається в змінну ch.

Рядки 11 і 13: при множенні на 2 значення змінної ch перетворюється в ціле число 66. При складені з величиною змінної f1 отримане в результаті число 132 перетворюється в число з плаваючою крапкою. Результат 197.00 перетворюється в число цілого типу і присвоюється змінній i.

Рядки 12 і 13: перед множенням на 2.0 значення змінної ch('B') перетворюється в число з плаваючою крапкою. Перед виконанням складення величина змінної i(197) перетворюється в число з плаваючою крапкою, а результат 329.00 присвоюється змінній f1.

Рядки 14 і 15: тут робиться спроба здійснити перетворення типів в порядку убування старшинства - змінна ch покладається рівною у порівнянні з більшим числом. Результати виявляються невтішними. Незалежно від переповнення і усічення, які мають місце, в результаті ми отримали код, відповідний якомусь недрукованому знаку.

Існує ще один вид перетворення типів. Для збереження точності обчислень при арифметичних операціях всі величини типу float перетворюються в дані типу double. Це істотно зменшує помилку округлення. Кінцевий результат перетворюється назад в число типу float, якщо це диктується відповідним оператором опису.

**Операція приведення**

Найкраще - це взагалі уникати перетворення типів, особливо в порядку убування рангу. Але іноді виявляється зручним застосовувати такі перетворення за умови, що ми ясно уявляємо сенс виконуваних дій. Перетворення типів, які ми обговорювали досі, виконувалися автоматично. Існує можливість точно вказувати тип даних, до якого необхідно привести величину. Цей спосіб називається **приведенням типів**, і використовується наступним чином: перед даною величиною в круглих дужках записується ім'я необхідного типу. Дужки і ім'я типу разом утворюють **операцію приведення**. У загальному вигляді вона записується так:

(тип)

де фактичне ім'я необхідного типу видається замість слова тип.

Розглянемо приклад:

int nice;

nice = 1.6+1.7;

nice = (int)1.6+(int)1.7;

У першому прикладі використовується автоматичне перетворення типів. Спочатку числа 1.6 і 1.7 складаються - результат дорівнює 3.3. Потім шляхом відкидання дробової частини отримане число перетворюється в 3 для узгодження з типом int змінної nice. У другому прикладі 1.6 і 1.7 перетворюються в цілі числа 1, так що зміній nice присвоюється значення, рівне 1 + 1, або 2.

**Особливості роботи з мовою Сі.** Ми не повинні змішувати типи; ось чому в деяких мовах це заборонено. Але бувають ситуації, коли це виявляється корисним. Філософія мови Сі полягає в тому, щоб не встановлювати бар'єрів на вашому шляху, але при цьому покласти на вас всю відповідальність за зловживання наданою свободою.

**Неявне перетворення типу**

Неявні перетворення типу виконуються головним чином для узгодження аргументів оператора або функції (якщо це можливо) зі значеннями, передбачуваними в цих операторах або функціях. Всі неявні перетворення типу, які можуть зустрітися, перераховані нижче (зліва вказується перетворений тип, а праворуч - список типів, в які він може бути перетворений):

char - int, short int, long int (Перетворення до значення з великим числом двійкових розрядів може включати, а може не включати розширення знакового розряду - це залежить від реалізації мови. Для елементів заданого набору знаків гарантується перетворення в невід'ємні цілі значення).

int - char, short int, long int ( перетворення до цілого більшої довжини включає розширення знакового розряду. Перетворення до цілого меншої довжини викликає відкидання зайвих старших розрядів). float, double, unsigned int (*інтерпретація* комбінації бітів у вигляді беззнакового цілого значення).

short int - аналогічно типу int.

long int - аналогічно типу int.

float - double, int, short int, long int (машинно-залежне перетворення, якщо значення, яке потрібно занадто велике, то результат невизначений).

double - float (перетворення з округленням і подальшим відкиданням зайвих розрядів), int, short int, long int.

**Арифметичне перетворення**

*Арифметичні оператори* мови Сі перетворять операнди до відповідних типів автоматично, якщо операнди не мали таких типів з самого початку. Схема перетворення, використовувана цими операторами, називається зазвичай **арифметичні перетворення**; ця схема може бути описана наступними правилами:

1. Претворити операнди типів char і short int до типу int ; Претворити операнди типу float до типу double.
2. Якщо хоча б один з операндів має тип double, то й інший операнд перетворюється в тип double (якщо він іншого типу); результат має тип double.
3. Якщо хоча б один з операнд має тип long, то й інший операнд перетворюється в тип long (якщо він іншого типу); результат має тип long.
4. Якщо хоча б один з операндів має тип unsigned, то й інший операнд перетворюється в тип unsigned (якщо його тип не unsigned); результат має тип unsigned.
5. Якщо ні один з випадків 1-4 не має місця, то обидва операнда повинні мати тип int ; такий же тип буде і у результату.

**Явне перетворення видів**

Вирази можуть бути перетворені з одного типу в інший явним зазначенням. Вираз E може бути явно перетворено до типу ім'я-типу за допомогою запису виду

(ім’я - типу) Е

Де ім’я типу показане в формі

покажчик-типу абстрактний-описатель

Абстрактний описатель аналогічний описателю, за винятком того, що він не містить визначеного або описуваного ідентифікатора. Сенс слів ім'я-типу, яку представляють в формі

Т абстрактний описатель

де Т є покажчиком типу, може бути визначений одним з таких способів:

* форма **абстрактного описателя** – сенс слів " Т **абстрактний описатель** ";
* пустй ( **абстрактний описатель** ) - абстрактний описатель типу Т ;
* \* ( **абстрактний описатель** ) - покажчик на тип Т ;
* абстрактний описатель ( ) - функція, яка повертає значення типу Т ;
* абстрактний описатель [ n ] – масив з n елементами типу Т, n – вираз з постійним значенням;

Наведемо приклади явного перетворення. Припустимо, що дано такі визначення і опис:

int i;

char \*pc, \*name;

char \*calloc( ), \*strcpy( );

тоді можна навести такі приклади явних перетворень типів:

(char) i - перетворює *значення* типу int в *значення* типу char.

pc=(char \*) 0777 - перетворює восьмеричний літер 0777 в значення покажчика на знак таким чином, що воно може бути присвоєно змінної "pc".

(emp \*) calloc(1,sizeof(emp)) - перетворює значення "знакового" покажчика, що повертається функцією calloc, в *значення покажчика* emp.

(void) strcpy(name,"gehani") - опускає значення, повернене функцією strcpy.

**Синтаксис типів**

Можна відзначити, що синтаксис типів в мові Сі нерегулярний і безладний, про що свідчать:

1. Труднощі створення форматерів.
2. Велика кількість помилок, що допускається навіть досвідченими програмістами.
3. Труднощі створення синтаксичних аналізаторів для трансляторів з мови Сі - не існує двох трансляторів з мови Сі, синтаксис вхідної мови для яких повністю збігався б.
4. Є три випадки, коли необхідно використовувати типи:
   * опису, що зв'язують тип з ім'ям, як наприклад, в разі опису (глобальних) змінних і формальних параметрів;
   * опису, що зв'язують значення (і тип) з ім'ям, як, наприклад, опису програм;
   * *приведення,* що забезпечують, наприклад, можливість трактування символів як цілих.