Структура на програма в C

Част 4 - оператори II

Иван Георгиев, Христо Иванов, Христо Стефанов

Технологично училище "Електронни системи",
Технически университет, София

16 март 2019 г.

Съдържание

Съдържание

Приоритет на операциите

- За да може един израз да бъде изпълнен, трябва да се определи в какъв ред да се изпълнят операциите в израза
- Подобно на математическите изрази, в C отделните операции имат приоритет на изпълнение
- В математиката операциите се изпълняват започвайки от операциите в най-вътрешните скоби на израза, след което се преминава към операциите в по-външните скоби и т.н. докато има скоби или операциите не свършат
- По същият начин се изпълняват и изразите в C

$$a \div c + d = (a \div c) + d$$

Фигура 1: Приоритет на извършване на математически операции

Асоциативност

- Когато имаме последователност от едни и същи операции възниква въпроса, коя от операциите да извършим първо
  - a^{b^c} се изпълнява като $a^{(b^c)}$ или като $(a^b)^c$?
- *Асоциативността* на една операция помага за определянето на реда на изпълняване на даден израз, когато в него има последователно прилагане на една и съща операция или операции със един и същ приоритет
  - *Дясно асоциативните* операции изпълняват първо всички операции в дясно от себе си
  - *Ляво асоциативните* операции изпълняват първо всички операции в ляво от себе си
- В математиката събиране, изваждане, умножение и деление са ляво асоциативни операции, а повдигането на степен е дясно асоциативна операция

Присвояване

- В езика C стойността на някоя променлива може да бъде променена чрез присвояване на нова стойност (т.е. стойността на променливата се замества с нова стойност)
- Операцията присвояване се отбелязва с *оператор за присвояване*

```
op1 = op2
```

Фрагмент 1: Синтаксис на оператор за присвояване

Оператор за присвояване

- За да може да бъде извършена операцията присвояване, трябва като първи операнд на оператора за присвояване да се посочи идентификатор на променлива, а като втори - произволен израз
  - Не е възможно да се присвояват нови стойности на константи/литерали, тъй като самия смисъл на константата/литерала забранява смяната на стойността
  - Присвояването на стойност на временна променлива няма практичен смисъл и затова е забранено
- Операнд, който може да участва като първи операнд в операция за присвояване, се нарича още *lvalue*, а такъв, който не може, *rvalue*
  - Такъв термин се въвежда, защото в C съществува оператор (и съответно израз), който може да създава временен идентификатор за някоя променлива

Оператор за присвояване

- Резултата от изпълнението на оператор за присвояване е `rvalue` със стойност новата стойност на променливата
- Присвояването е дясно асоциативна операция

```
{
    int a = 10;
    int b = 20;
    a = 5; // OK, a is 'lvalue'
    5 = a; // ERROR, 5 is not an 'lvalue', it is a 'rvalue'
    (a = 5) = 10; // ERROR, the expression '(a = 5)' is a 'rvalue'
    a = b = 40; // OK, assignment is right-associative operation
}
```

Фрагмент 2: Правилна и неправилна употреба на оператор за присвояване

Съдържание

Аритметични оператори

- Езикът C позволява извършването на аритметични операции, използвайки *аритметични оператори*

Име на оператора	Синтаксис[1]	Операция
Събиране	`op1 + op2`	Събиране на `op1` и `op2`
Изваждане	`op1 - op2`	Изваждане на `op1` и `op2`
Умножение	`op1 * op2`	Умножение на `op1` и `op2`
Деление	`op1 / op2`	Деление на `op1` и `op2`
Остатък от деление	`op1 % op2`	Остатък от деление на `op1` и `op2`

Таблица 1: Бинарни аритметични оператори в C

[1] `op1` и `op2`, могат да са rvalue или lvalue

Бинарни аритметични оператори

- Бинарните аритметични операции работят винаги върху данни от един и същ тип и връщат *rvalue* стойност от същия тип
- Ако се подадат операнди от различен тип, се извършва *неявно преобразуване на типа* (*implicit type cast/conversion*) на някой от операндите, за да се уеднаквят типовете
- Неявното преобразуване преобразува типа на операндата с по-малък обхват от стойности към типа на операндата с по-голям обхват от стойности
- Например:

```
{
    int a = 4;
    long int b = 7;
    long int c = a + b; // OK, 'a' is implicitly converted to 'long int'
    float d = 57.8;
    float e = d + c; // OK, 'c' is implicitly converted to 'float'
}
```

Фрагмент 3: Неявно преобразуване на типове при бинарни аритметични оператори

Унарни аритметични оператори (1/2)

- *Операторът за смяна на знак* сменя знака на стойността на своя операнд и връща резултата като rvalue.
- *Унарният плюс* връща стойността на операнда непроменена като rvalue

Име на оператора	Синтаксис[2]
Смяна на знак (*унарен минус*)	-op1
Унарен плюс	+op1

Таблица 2: Унарни аритметични оператори в C (1/2)

[2] op1 може да е rvalue или lvalue

Унарни аритметични оператори (2/2)

- Съществуват поредици от операции, коти много често се използват при писането на програми
- Езикът C предоставя операции, които изпълняват някоя от тези поредици като една операция, с цел по-малко писане на код

Име на оператора	Синтаксис[3]
Префиксно инкрементиране	`++op1`
Постфиксно инкрементиране	`op1++`
Префиксно декрементиране	`--op1`
Постфиксно декрементиране	`op1--`

Таблица 3: Унарни аритметични оператори в C (2/2)

[3] op1 може да е само lvalue

Унарни аритметични оператори

- *Операторът за префиксно инкрементиране* увеличава стойността на своя операнд с 1 и връща *новата* стойност на операнда като rvalue. Операндът е задължително *lvalue*.
- *Операторът за постфиксно инкрементиране* увеличава стойността на своя операнд с 1 и връща *старата* стойност на операнда като rvalue. Операндът е задължително *lvalue*.
- *Операторът за префиксно декрементиране* намалява стойността на своя операнд с 1 и връща *новата* стойност на операнда като rvalue. Операндът е задължително *lvalue*.
- *Операторът за постфиксно декрементиране* намалява стойността на своя операнд с 1 и връща *старата* стойност на операнда като rvalue. Операндът е задължително *lvalue*.

Унарни аритметични оператори

```
{
    int a = 1;
    int b = 1;
    b = ++a; // b is now 2, a is now 2
    b = a++; // b is still 2, a is now 3
    b = -a; // b is now -3, a is unchanged
    a = b--; // a is now -3, b is now -4
    a++; // a is now -2
    b = +a; // b is now -2, a is unchanged
}
```

Фрагмент 4: Работа с унарни аритметични оператори

Съдържание

Логически оператори

- *Логическите оператори* изпълняват логически операции за истинност върху своите операнди
- При изпълнение на логическите операции за истинна стойност се приема всяка стойност на операнд различна от 0, а за лъжлива стойност, стойност на операнд равна на 0
- Резултата от операциите са или истина или лъжа. При истина винаги се връща стойност 1, а при лъжа - 0
- При изпълнение се изчислява първо op1 и само ако не може да се определи резултата на операцията от op1, се изчислява и op2

Име на оператора	Синтаксис[4]	Операция
Логическо И	`op1 && op2`	Логическо И
Логическо ИЛИ	`op1 || op2`	Логическо ИЛИ
Логическо НЕ	`!op1`	Логическо НЕ

Таблица 4: Логически оператори

[4] op1 и op2 могат да са lvalue или rvalue

Логически оператори

```
{
    int a = 10;
    int b = 0;
    int c = 10 || 3; // 'c' is now 1
    int d = 0 && 10; // 'd' is now 0
    int e = !c; // 'e' is now 0
    int f = c || a++; // 'f' is now 1, 'a' is still 10
    int d = b || a++; // 'd' is now 1, 'a' is now 11
}
```

Фрагмент 5: Работа с логически операции

Съдържание

Сравняващи (релационни) оператори

- *Сравняващите оператори* (*релационни оператори*) са бинарни оператори, които сравняват два операнда
- Аналогично на бинарните аритметични оператори, ако типовете на операндите се различават, се извършва *неявно преобразуване на типовете* на операндите
- Ако релацията между операндите важи, резултатът е rvalue със стойност 1, иначе е rvalue със стойност 0

Сравняващи (релационни) оператори

Име на оператора	Синтаксис[5]	Върната `rvalue` стойност
Равно на	`op1 == op2`	1, ако са равни, иначе 0
Различно от	`op1 != op2`	1, ако не са равни, иначе 0
По-голямо от	`op1 > op2`	1, ако стойността на op1 е по-голяма, иначе 0
По-малко от	`op1 < op2`	1, ако стойността на op1 е по-малка, иначе 0
По-голямо или равно на	`op1 >= op2`	1, ако стойността на op1 е по-голяма или равна на op2, иначе 0
По-малко или равно на	`op1 <= op2`	1, ако стойността на op1 е по-малка или равна на op2, иначе 0

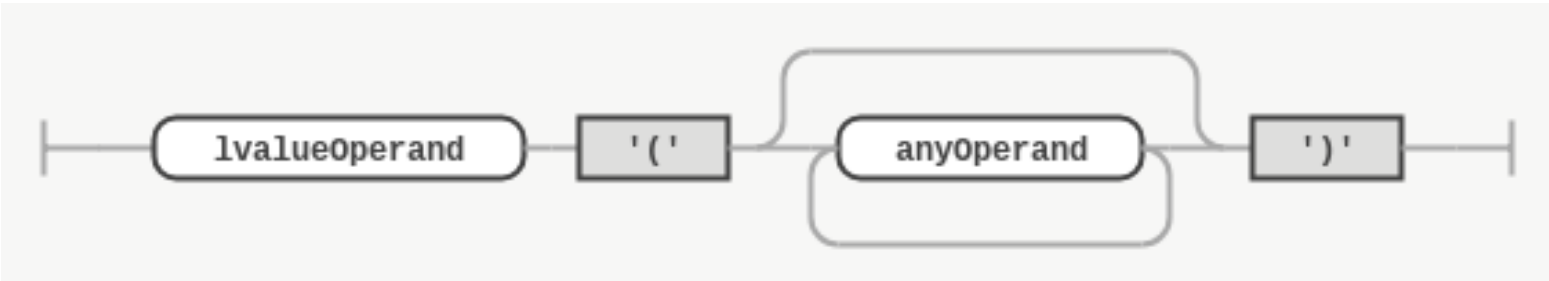
Таблица 5: Сравняващи (релационни) оператори

[5] op1 и op2 могат да са lvalue или rvalue

Съдържание

Оператор за извикване на функция

- *Операторът за извикване на функция* изпълнява функция, посочена от lvalue операнда, с параметри с първоначални стойности равни на стойностите на останалите подадени операнди (наречени *аргументи*)
- Броят на аргументите трябва да е точно толкова, колкото броят на параметрите на функцията



Фигура 2: Синтактична диаграма на оператор за извикване на функция

Оператор за извикване на функция

- Преди изпълнението на функцията, стойността на поредния подаден аргумент се използва за първоначална стойност на поредния параметър на функцията
- Типът на поредния аргумент трябва да съвпада с типа на поредния параметър
- Ако типът не съвпада, се извършва неявно преобразуване на типа на аргумента към типа на параметъра, стига да е възможно такова преобразуване
- Върнатата стойност от операцията извикване на функция е резултата от изпълнението на функцията. Стойността е rvalue стойност

Оператор за извикване на функция

```c
float sum(int a, float b) {
    return a + b;
}

int main() {
    int x = 10;
    float y = 30;
    char z = 10;
    float r = 0;
    sum(x, y); // OK, the number of arguments and their types
               // match the number of parameters and their types
    sum(x); // ERROR, 'sum' requires 2 arguments
    sum(x, y, z); // ERROR, 'sum' requires 2 arguments
    sum(z, x); // OK, the value of 'z' is converted to 'int'
    r = sum(z, x) + 10; // OK
}
```

Фрагмент 6: Работа с оператор за извикване на функции