УРОК 02

1

Добре дошли в урок 2 от нашия курс по програмиране на C! В този урок ще обсъдим типовете стойности и тяхното представяне в компютъра и в езика C.

Първо, нека поговорим за двете категории стойности : примитивни стойности и съставни стойности. Примитивните стойности са основните градивни елементи на данните в C, докато съставните стойности се създават чрез комбиниране на примитивни стойности.

C поддържа няколко примитивни типа данни, които включват:

Цели числа: Това са цели числа без десетична запетая. C има четири типа цели числа, които са char, short, int и long. Размерът на тези цели числа може да варира в зависимост от системата, върху която работите, но всички те представляват цели числа.

Числа с плаваща запетая: Това са числа с десетична запетая. C има два типа с плаваща запетая, които са float и double. Плаващите са обикновено 32-битови, докато двойните са 64-битови.

Знаци: Това са единични буквено-цифрови символи или специални знаци, затворени в единични кавички. C има тип char за представяне на знаци.

Boolean: Това е тип, който може да приема само две стойности: true или false. В C boolean не е примитивен тип, но може да се дефинира с помощта на типа enum.

Сега нека обсъдим как тези стойности са представени в компютъра. Компютрите използват двоична система, което означава, че разбират само две стойности: 0 и 1. Следователно всички данни в компютъра се представят като последователност от битове (0 и 1). Начинът, по който стойностите са представени в двоичен вид, зависи от техния тип.

За целите числа най-често срещаният начин за представянето им е като допълнение към две. Допълнението на две е начин за представяне на отрицателни числа, като се използва същият брой битове като положителните числа. Например 8-битово цяло число може да представлява стойности от -128 до 127.

Числата с плаваща запетая се представят с помощта на стандартизиран формат, наречен IEEE 754. Този формат представя числото под формата на знак-величина с експонента и мантиса.

Символите се представят с помощта на ASCII (Американски стандартен код за обмен на информация), който присвоява уникален 8-битов код на всеки знак.

Булевите стойности се представят с помощта на един бит, където 0 представлява false и 1 представлява true.

Нека сега да разгледаме някои реални примери на код, които илюстрират тези концепции:

```
#include <stdio.h>
int main() {
char myChar = 'a';
int myInt = 42;
float myFloat = 3.14;
double myDouble = 3.14159265359;
_Bool myBool = 1;
printf("myChar: %c\n", myChar);
printf("myInt: %d\n", myInt);
printf("myFloat: %f\n", myFloat);
printf("myDouble: %lf\n", myDouble);
printf("myBool: %d\n", myBool);
return 0;
}
```

В този пример ние дефинираме променливи от различни типове и отпечатваме техните стойности на конзолата с помощта на функцията printf. Можем да видим как различните типове са представени в изхода.

В заключение, разбирането на типовете стойности и тяхното представяне в компютъра и в C е от съществено значение за писането на ефективни и ефективни програми. Уверете се, че сте запознати с тези концепции, преди да преминете към по-напредналите теми в C програмирането.

2

Добре дошли в част 2 на урок 2 от нашия курс по програмиране на C! В този урок ще обсъдим двоичните, осмичните и шестнадесетичните бройни системи.

В C ние обикновено използваме десетичната бройна система, която използва 10 цифри (0 до 9) за представяне на числа. В компютърните науки обаче често използваме други бройни системи, като двоична, осмична и шестнадесетична, за представяне на данни.

Двоична бройна система:

Двоичната бройна система използва само две цифри, 0 и 1, за представяне на числа. Той е в основата на всички компютърни операции и се използва широко в цифровата електроника. Всяка цифра в двоично число представлява степен на 2. Например, двоичното число 1011 представлява десетичното число (1 x 2^3) + (0 x 2^2) + (1 x 2^1) + (1 x 2^0) = 11.

В C можем да представим двоични числа, като им поставим префикс 0b или 0B. Например:

```
int binaryNumber = 0b1011; // binaryNumber is 11 in decimal
```

Осмична бройна система:

Осмичната бройна система използва 8 цифри (0 до 7) за представяне на числа. Всяка цифра в осмично число представлява степен на 8. Например осмичното число 27 представлява десетичното число (2 x 8^1) + (7 x 8^0) = 23.

В C можем да представим осмични числа, като им поставим префикс 0. Например:

```
int octalNumber = 031; // octalNumber is 25 in decimal
```

Шестнадесетична бройна система:

Шестнадесетичната бройна система използва 16 цифри (0 до 9 и A до F) за представяне на числа. Всяка цифра в шестнадесетично число представлява степен на 16. Например шестнадесетичното число 1A представлява десетичното число (1 x 16^1) + (10 x 16^0) = 26.

В C можем да представим шестнадесетични числа, като им поставим префикс 0x или 0X. Например:

```
int hexadecimalNumber = 0x1A; // hexadecimalNumber is 26 in decimal
```

Сега нека да разгледаме някои примери за кодове, които използват двоични, осмични и шестнадесетични числа:

```c
#include <stdio.h>
int main() {
int binaryNumber = 0b1011;
int octalNumber = 031;
int hexadecimalNumber = 0x1A;
printf("binaryNumber: %d\n", binaryNumber);
printf("octalNumber: %d\n", octalNumber);
printf("hexadecimalNumber: %d\n", hexadecimalNumber);
return 0;
}
```

В този пример ние дефинираме променливи с помощта на двоични, осмични и шестнадесетични числа и отпечатваме техните стойности на конзолата с помощта на функцията printf.

В заключение, разбирането на двоичните, осмичните и шестнадесетичните бройни системи е важно в компютърните науки и програмирането. Те често се използват за представяне на данни и са от съществено значение при програмирането на ниско ниво. В C можем да използваме префикси за представяне на числа в тези системи.

3

Добре дошли в урок 2 от нашия курс по програмиране на C! В този урок ще обсъдим как да извлечем число от неговия запис в позиционната бройна система и обратно.

Извличане на число от неговия запис:

За да извлечем число от неговия запис в позиционната бройна система, трябва да умножим всяка цифра по съответната степен на основата и след това да сумираме резултатите. Например, нека разгледаме числото 347 при основа 10. Цифрата 3 е на мястото на стотиците, цифрата 4 е на мястото на десетиците, а цифрата 7 е на мястото на единиците. Така че можем да изчислим стойността на числото, както следва:

347 = (3 x 10^2) + (4 x 10^1) + (7 x 10^0)
= 300 + 40 + 7
347

В C можем да извлечем число от нотацията му с помощта на цикъл и някои основни аритметични действия. Ето един пример:

```
#include <stdio.h>
#include <math.h>
int main() {
int number = 347;
int base = 10;
int result = 0;
int placeValue = 0;
while (number > 0) {
int digit = number % 10;
result += digit * pow(base, placeValue);
placeValue++;
number /= 10;
}

return 0;
}
```

В този пример използваме цикъл while, за да извлечем цифрите на числото и да изчислим резултата, като умножим всяка цифра по съответната степен на основата и след това сумираме резултатите.

Преобразуване на число в неговия запис:

За да преобразуваме число в неговия запис в позиционната бройна система, трябва многократно да разделяме числото на основата и да записваме остатъка на всяка стъпка. Остатъците ще образуват цифрите на числото в обратен ред. Например, нека разгледаме числото 347 в основа 10. Можем да го преобразуваме в основа 2, както следва:

```
347 / 2 = 173 remainder 1
173 / 2 = 86 remainder 0
86 / 2 = 43 remainder 0
```

```
43 / 2 = 21 remainder 1
21 / 2 = 10 remainder 1
10 / 2 = 5 remainder 0
5 / 2 = 2 remainder 1
2 / 2 = 1 remainder 0
1 / 2 = 0 remainder 1
```

Остатъците в обратен ред са 101011011, което е двоичното представяне на 347.

В C можем да преобразуваме число в неговата нотация с помощта на цикъл и оператора modulo. Ето един пример:

```
#include <stdio.h>
int main() {
int number = 347;
int base = 2;
int result = 0;
int placeValue = 1;
while (number > 0) {
int digit = number % base;
result += digit * placeValue;
placeValue *= 10;
number /= base;
}

return 0;
}
```

В този пример използваме цикъл while, за да разделим числото на основата и да запишем остатъците. След това използваме остатъците, за да изчислим резултата, като умножим всяка цифра по съответната стойност на място и след това сумираме резултатите.

В заключение, разбирането как да се извлече число от неговия запис в позиционната бройна система и обратно е от съществено значение за програмирането. Можем да използваме цикли и основна аритметика.

4

Добре дошли в урок 2 от нашия курс по програмиране на C! В този урок ще обсъдим **имената и свойствата на стандартните числови типове в C.**

C предоставя няколко вградени типа числа, които могат да бъдат класифицирани в две категории: цели числа и типове с плаваща запетая.

Целочислени типове: C предоставя няколко цели числа, които се различават по своя обхват и количеството памет, което заемат. Следната таблица обобщава имената и свойствата на стандартните цели числа в C:

Тип име	Обхват	Размер (байтове)
char	-128 до 127 или 0 до 255 (ако е без знак)	1
short	-32 768 до 32 767 или 0 до 65 535 (ако е без знак)	2
int	-2,147,483,648 до 2,147,483,647 или 0 до 4,294,967,295 (ако е без знак)	4
long	-2,147,483,648 до 2,147,483,647 или 0 до 4,294,967,295 (ако е без знак)	4 или 8
long long	-9,223,372,036,854,775,808 до 9,223,372,036,854,775,807 или 0 до 18,446,744,073,709,551,615 (ако е без знак)	8

Имайте предвид, че **точният диапазон** и размер на всеки тип цяло число може да **зависи от системата и компилатора**, които използвате.

Типове с плаваща запетая: C предоставя два типа с плаваща запетая: float и double. Тези типове представляват реални числа с различни нива на точност. Следната таблица обобщава имената и свойствата на стандартните типове с плаваща запетая в C:

Тип име	Обхват	Точност (цифри)
float	1.2E-38 до 3.4E+38	6

double	2.3E-308 до 1.7E+308	15

Имайте предвид, че точният обхват и точност на всеки тип с плаваща запетая може да зависи от системата и компилатора, които използвате.

В C можем да декларираме променливи от тези типове, като използваме следния синтаксис:

```
// integer types
char c;
short s;
int i;
long l;
long long ll;
// floating-point types
float f;
double d;
```

Можем също да използваме sizeof оператора, за да определим размера на всеки тип в байтове:

```
printf("Size of char: %lu bytes\n", sizeof(char));
printf("Size of short: %lu bytes\n", sizeof(short));
printf("Size of int: %lu bytes\n", sizeof(int));
printf("Size of long: %lu bytes\n", sizeof(long));
printf("Size of long long: %lu bytes\n", sizeof(long long));
printf("Size of float: %lu bytes\n", sizeof(float));
printf("Size of double: %lu bytes\n", sizeof(double));
```

В заключение, разбирането на имената и свойствата на стандартните типове числа е от решаващо значение при програмирането. Това ни позволява да изберем подходящия тип за дадена задача и да гарантираме, че нашите програми са ефективни и точни.

5

Добре дошли в част 05 на урок 2 от нашия курс по програмиране на C! В този урок ще обсъдим как да задаваме променливи в C.

Променливите се използват за съхраняване на данни в паметта. В C можем да декларираме променливи от различни типове, както обсъдихме в предишната подтема. След като декларираме променлива, можем да зададем нейната стойност с помощта на оператора за присвояване =.

Нека разгледаме един пример. Да предположим, че искаме да съхраним числото 42 в целочислена променлива, наречена num. Можем да декларираме и зададем променливата, както следва:

```
int num; // declaration
num = 42; // setting the value
```

Като алтернатива можем да декларираме и зададем променливата в един ред:

```
int num = 42; // declaration and setting the value
```

Можем също да зададем стойността на променлива въз основа на нейната предишна стойност. Да предположим например, че искаме да увеличим стойността на num с 1. Можем да го направим, като използваме следния код:

```
num = num + 1; // incrementing the value by 1
```

Или можем да използваме съкратената нотация:

```
num++; // incrementing the value by 1 using shorthand notation
```

По същия начин можем да намалим стойността на променлива с 1, като използваме следния код:

```
num = num - 1; // decrementing the value by 1
```

Или можем да използваме съкратената нотация:

```
num--; // decrementing the value by 1 using shorthand notation
```

Можем също да зададем стойността на променлива въз основа на стойностите на други променливи. Да предположим например, че имаме две целочислени променливи xи y, и искаме да зададем стойността на питна сумата от xи y. Можем да го направим, като използваме следния код:

```
int x = 10;
int y = 20;
int num = x + y; // setting the value based on the values of other variables
```

В заключение, задаването на променливи е фундаментална концепция в програмирането. Тя ни позволява да съхраняваме данни в паметта и да ги манипулираме според нуждите. Като разберем как да задаваме променливи в C, можем да пишем програми, които са ефективни и точни.

6

Добре дошли в урок 2, част 06 от нашия курс по програмиране на C! В този урок ще обсъдим **операции и изрази в C**.

В C **операцията** е **математическа** или **логическа** операция, която се извършва върху един или повече **операнди**, за да се получи резултат. Например, операцията събиране взема два операнда и произвежда тяхната сума като резултат.

Изразът в C е **комбинация** от **операнди** и **операции**, която произвежда стойност. Например 2 + 3 е израз, който произвежда стойността 5.

Нека разгледаме някои примери за операции и изрази в C.

```
int x = 10;
int y = 20;
int z;
z = x + y; // addition operation
printf("The sum of x and y is: %d\n", z); // expression
z = x - y; // subtraction operation
printf("The difference between x and y is: %d\n", z); // expression
z = x * y; // multiplication operation
printf("The product of x and y is: %d\n", z); // expression
z = y / x; // division operation
```

```
printf("The quotient of y divided by x is: %d\n", z); // expression
z = y % x; // modulus operation
printf("The remainder of y divided by x is: %d\n", z); // expression
int a = 5;
int b = 2;
float c;
c = (float) a / b; // type casting operation
printf("The result of a divided by b is: %f\n", c); // expression
```

7

В горния пример сме използвали различни аритметични операции като събиране, изваждане, умножение, деление и модул. Ние също използвахме операцията за преобразуване на типа, за да преобразуваме целочислената променлива ав променлива с плаваща задна буква.

Можем също да използваме логически операции като И (&&), ИЛИ (||) и НЕ (!) в C. Тези операции обикновено се използват в условни изрази за вземане на решения въз основа на определени условия.

```
int age = 25;
int weight = 60;
if (age >= 18 && weight < 100) {
printf("You are eligible to donate blood.\n"); // conditional statement using logical AND
}
if (age < 18 || weight >= 100) {
printf("You are not eligible to donate blood.\n"); // conditional statement using logical OR
}
if (!(age < 18)) {
printf("You are old enough to vote.\n"); // conditional statement using logical NOT
}
```

В горния пример използвахме логическите операции И (&&), ИЛИ (||) и НЕ (!) в условни изрази, за да определим допустимостта за кръводаряване и гласуване.

В заключение, операциите и изразите са важни понятия в програмирането на C. Те ни позволяват да извършваме математически и логически операции с променливи и да вземаме решения въз основа на определени условия. Като разберем как да използваме операции и изрази в C, можем да пишем програми, които са ефективни и точни.

8

В този раздел ще обсъдим **конструирането на изрази** в програмирането на C, реда на изчисление, целочислени и реални изчисления, подравняване и преобразуване на числа по тип, псевдооперации и други функции.

Изразите в C програмирането са съставени от операнди и оператори.

Операндите могат да бъдат променливи, константи или извиквания на функции.

Операторите могат да бъдат аритметични, логически, релационни или побитови. Операндите и операторите се комбинират, за да образуват изрази, които се оценяват, за да произведат стойност.

Редът на изчисленията в програмирането на C следва правилата за приоритет на операторите и асоциативност. Приоритетът на операторите определя кои оператори се оценяват първи, а асоциативността определя реда на оценяване за оператори със същия приоритет.

Целочислените изчисления в програмирането на C се извършват с помощта на цели числа, които са цели числа без дробни части. Реалните изчисления се извършват с помощта на числа с плаваща запетая, които са числа с дробни части.

Подравняването и преобразуването на числата по тип е важна характеристика в C програмирането. C предоставя различни типове данни за представяне на различни видове стойности. Например, целите числа могат да бъдат представени с помощта на типове данни char, short, int, long и long long, а числата с плаваща запетая могат да бъдат представени с помощта на типове данни float, double и long double. Когато операндите от различни типове се комбинират в израз, C извършва автоматично преобразуване на типа, за да гарантира, че операндите са от един и същи тип.

Програмирането на C също така предоставя псевдооперации, които са специални функции, които не съществуват в стандартната библиотека, но са вградени в самия език. Примери за псевдооперации включват sizeof, който връща размера на даден тип данни в байтове, и троичен оператор ?:, който е съкратен начин за писане на оператор if-else.

Нека да разгледаме някои примери за изрази в C програмирането:

```
int x = 5, y = 10;
float z = 3.14;
// Arithmetic expressions
int sum = x + y;
int difference = x - y;
```

```
float product = x * z;
float quotient = y / z;
// Logical expressions
int and_result = x && y;
int or_result = x || y;
int not_result = !x;
// Relational expressions
// По-малко от int less_than = x < y;

// По-голямо от int larger_than = x > y;

// По-малко или равно на int less_than_or_equal_to = x <= y;

// По-голямо или равно на int larger_than_or_equal_to = x >= y;

// Равенство int equal = x == y;

// Неравенство int not_equal = x != y;

// Побитови изрази int bitwise_and = x & y; int побитово_или = x | y; int bitwise_xor = x ^ y; int
побитово_допълнение = ~x;
https://www.geeksforgeeks.org/bitwise-operators-in-c-cpp/

// Псевдооперации int size_of_x = sizeof(x); int max_value_of_int = INT_MAX; int ternary_result = (x > y)? x : y;
```

Обърнете внимание, че в горните примери се извършва автоматично преобразуване на типовете, ако е необходимо, за да се гарантира, че операндите са от един и същ тип. Например, в израза `x * z`, `x` се преобразува автоматично в float, за да може умножението да се извърши с два float операнда.

В заключение, разбирането на конструкцията на изразите, реда на изчисленията, целочислените и реалните изчисления, подравняването и преобразуването на числата по тип, псевдоопериациите и други характеристики е от решаващо значение за писането на ефективни и ефикасни програми на C. Като овладеете тези понятия, ще можете да пишете сложни изрази и да извършвате изчисления с лекота.

9

В програмирането на **С входните и изходните операции** са от съществено значение за създаването на полезни програми. Стандартните входни и изходни потоци са stdin и stdout съответно, а стандартният поток за грешки е stderr. Тези потоци позволяват на програмите да четат въведени данни от потребителя, да показват изход на потребителя и да докладват за грешки.

За да използвате тези потоци, трябва да включите съответните заглавни файлове. Заглавният stdio.hфайл предоставя необходимите функции за извършване на входни и изходни операции. Ето някои от най-често използваните функции:

scanf( ): чете въведени данни от stdin според определен низ от формат
printf( ): записва изход в stdout съответствие с определен низ за форматиране
fprintf( ): записва изход в определен файлов поток според определен низ на формат
fgets( ): чете входен ред от stdin и го съхранява в масив от знаци
fputs( ): записва низ в определен файлов поток
puts( ): записва низ към tdout

Ето примерна програма, която демонстрира използването на стандартен вход и изход:

```
#include <stdio.h>
int main() {
int num1, num2, sum;
printf("Enter two numbers separated by a space: ");
scanf("%d %d", &num1, &num2);
sum = num1 + num2;
printf("The sum of %d and %d is %d\n", num1, num2, sum);
return 0;
}
```

В тази програма printf() функцията се използва за показване на подкана към потребителя и функцията scanf() се използва за четене на въведени от потребителя данни. Входът се съхранява в променливите num1и num2. След това сумата от двете числа се изчислява и показва на потребителя с помощта на printf() функцията.

Можете също така да пренасочите входа и изхода с помощта на командния ред. Например, за да пренасочите стандартния вход от файл, наречен, input.txt и стандартния изход към файл, наречен output.txt, можете да използвате следната команда:

```
./my_program < input.txt > output.txt
```

В допълнение към стандартните входни и изходни функции, стандартната библиотека C предоставя много други средства за работа с файлове, разпределяне на памет, манипулиране на низове и други често срещани програмни задачи. Като използвате тези библиотечни функции, можете да спестите време и усилия и да пишете по-ефективни и ефективни програми.

10

В езика за програмиране C **програмите за филтриране** могат да бъдат написани с помощта на стандартните библиотечни функции за вход и изход. Основната функция за четене на вход е **fgets()**, която чете ред текст от стандартен вход и го съхранява в масив от знаци. Основната функция за изход е **printf()**, която записва форматиран изход в стандартен изход.

Ето пример за филтърна програма, която чете редове текст от стандартен вход, премахва всички гласни от всеки ред и записва получените редове в стандартен изход:

```
#include <stdio.h>
#include <string.h>
int main() {
char line[1000];

while (fgets(line, sizeof(line), stdin)) {

for (int i = 0; i < strlen(line); i++) {
if (line[i] == 'a' || line[i] == 'e' || line[i] == 'i' || line[i] == 'o' || line[i] == 'u'
|| line[i] == 'A' || line[i] == 'E' || line[i] == 'I' || line[i] == 'O' || line[i] == 'U') {
continue;
}
printf("%c", line[i]);
}
```

```
}
return 0;
}
```

В тази програма използваме масив от знаци, line за да съхраняваме всеки ред текст, прочетен от стандартен вход, използвайки fgets(). След това преминаваме през всеки знак в реда, като проверяваме дали е гласна. Ако е, продължаваме към следващия знак. Ако не е, използваме printf() за запис на знака в стандартния изход.

Програмите за филтриране могат да се използват за широк набор от задачи, като обработка на текст, анализ на данни и манипулиране на файлове. Те често се използват заедно с други програми за създаване на тръбопроводи, където изходът на една програма се използва като вход за друга програма.

Филтриращите програми са програми, които вземат входни данни от стандартния вход, обработват ги и след това извеждат резултата в стандартен изход. Те се наричат филтриращи програми, защото филтрират данни: те приемат данни, правят нещо с тях и извеждат модифицираните данни.

Програмите за филтриране могат да бъдат много полезни в различни контексти. Например, те могат да се използват за модифициране на текстови файлове, извършване на трансформации на данни или извличане на специфична информация от входни данни.

Нека да разгледаме пример за филтърна програма. Следната програма чете вход от стандартен вход, брои броя на редовете във входа и извежда резултата на стандартен изход:

```
#include <stdio.h>
int main() {
int c;
int lines = 0;
while ((c = getchar()) != EOF) {
if (c == '\n') {
lines++;
}
}
printf("Number of lines: %d\n", lines);
```

```
return 0;
}
```

Тази програма използва getchar() функцията за четене на въведен символ по знак от стандартния вход. Той отчита броя на редовете, като проверява дали всеки знак е символ за нов ред ('\n'). Ако бъде намерен знак за нов ред, броят на редовете се увеличава.

След като програмата приключи обработката на входа, тя извежда броя на редовете към стандартния изход, като използва функцията printf().

За да използвате тази програма, обикновено трябва да подадете вход към нея от друга програма или файл. Например, ако имате извикан файл input.txt, в който искате да преброите броя на редовете, можете да изпълните следната команда:

```
$ cat input.txt | ./linecount
```

Тази команда използва cat командата за извеждане на съдържанието на input.txt стандартен изход и след това прехвърля този изход към linecount програмата. Програмата linecount чете входа от стандартния вход, брои броя на редовете и извежда резултата на стандартния изход.

Програмите за филтриране могат да бъдат доста мощни, тъй като ви позволяват лесно да обработвате големи количества данни, без да се налага да пишете сложен файлов I/O код. С малко креативност можете да използвате филтриращи програми за решаване на голямо разнообразие от проблеми.