Fundamentals of Computer Programming

C Programming
2. Expressions

Contents

The Basic Data Types

- `char`
- `int`
- `float`
- `double`
- `void`

Modifying the Basic Types

- `signed`
- `unsigned`
- `long`
- `short`

Table 2-1. All Data Types Defined by the C Standard

Type	Typical Size in Bits	Minimal Range
char	8	–127 to 127
unsigned char	8	0 to 255
signed char	8	–127 to 127
int	16 or 32	–32,767 to 32,767
unsigned int	16 or 32	0 to 65,535
signed int	16 or 32	Same as **int**
short int	16	–32,767 to 32,767
unsigned short int	16	0 to 65,535
signed short int	16	Same as **short int**

Table 2-1. All Data Types Defined by the C Standard (cont.)

long int	32	–2,147,483,647 to 2,147,483,647
long long int	64	$-(2^{63}-1)$ to $2^{63}-1$ (Added by C99)
signed long int	32	Same as **long int**
unsigned long int	32	0 to 4,294,967,295
unsigned long long int	64	$2^{64}-1$ (Added by C99)
float	32	1E –37 to 1E+37 with six digits of precision
double	64	1E –37 to 1E+37 with ten digits of precision
long double	80	1E –37 to 1E+37 with ten digits of precision

Identifier Names

- The names of variables, functions, labels, and various other user-defined items are called *identifiers*.
- The length of these identifiers can vary from one to several characters.
- The first character must be a letter or an underscore, and subsequent characters must be either letters, digits, or underscores.

Variables

type variable_list;

```
int i, j, l;
short int si;
unsigned int ui;
double balance, profit, loss;
```

Where Variables Are Declared

- Variables can be declared in three places: inside functions, in the definition of function parameters, and outside of all functions.
- These positions correspond to *local variables*, *formal parameters*, and *global variables*, respectively.

Local Variables

- Variables that are declared inside a function are called *local variables*.
- Local variables exist only while the *block of code* in which they are declared is executing.

```
void func1(void)
{
   int x;

   x = 10;
}

void func2(void)
{
   int x;

   x = -199;
}
```

Formal Parameters

- If a function is to use *arguments*, it must declare variables that will accept the values of the arguments.
- These variables are called the *formal parameters* of the function.
- They behave like any other local variables inside the function.

Global Variables

- *Global variables* are known throughout the program and may be used by any piece of code.
- Storage for global variables is in a fixed region of memory set aside for this purpose by the compiler.

```c
#include <stdio.h>
int count;  /* count is global  */

void func1(void);
void func2(void);

int main(void)
{
  count = 100;
  func1();

  return 0;
}

void func1(void)
{
  int temp;

  temp = count;
  func2();
  printf("count is %d", count); /* will print 100 */
}

void func2(void)
{
  int count;

  for(count=1; count<10; count++)
    putchar('.');
}
```

The Four C Scopes

- File scope
- Block scope
- Function prototype Scope
- Function scope

Variable Initializations

type variable_name = constant;

```
char ch = 'a';
int first = 0;
double balance = 123.23;
```

Constants

- Constants refer to fixed values that the program may not alter.

Data Type	Constant Examples
int	1 123 21000 –234
long int	35000L –34L
unsigned int	10000U 987u 40000U
float	123.23F 4.34e–3f
double	123.23 1.0 –0.9876324
long double	1001.2L

Operators

- Arithmetic
- Relational
- Logical
- Bitwise

The Assignment Operator

- `variable_name = expression;`
- When variables of one type are mixed with variables of another type, a *type conversion* will occur.

```
int x;
char ch;
float  f;

void func(void)
{
  ch = x;          /* line 1 */
  x = f;           /* line 2 */
  f = ch;          /* line 3 */
  f = x;           /* line 4 */
}
```

Target Type	Expression Type	Possible Info Loss
signed char	char	If value > 127, target is negative
char	short int	High-order 8 bits
char	int (16 bits)	High-order 8 bits
char	int (32 bits)	High-order 24 bits
char	long int	High-order 24 bits
short int	int (16 bits)	None
short int	int (32 bits)	High-order 16 bits
int (16 bits)	long int	High-order 16 bits
int (32 bits)	long int	None
long int (32 bits)	long long int (64 bits)	High-order 32 bits (applies to C99 only)
int	float	Fractional part and possibly more
float	double	Precision, result rounded
double	long double	Precision, result rounded

Compound Assignments

- ```
  x += 10;
  ```
- The operator += tells the compiler to assign to x the value of x plus 10.
- `var = var operator expression`
  can be rewritten as
  `var operator = expression`
  ```

Arithmetic Operators

- %: the remainder of an integer division. However, you cannot use it on floating-point types.

```
int x, y;

x = 5;
y = 2;

printf("%d ", x/y);  /* will display 2 */
printf(''%d ", x%y);  /* will display 1, the remainder of
                         the integer division */
```

```
x = 1;
y = 2;

printf("%d %d", x/y, x%y); /* will display 0 1 */
```

Arithmetic Operators

Operator	Action
–	Subtraction, also unary minus
+	Addition
*	Multiplication
/	Division
%	Modulus
––	Decrement
++	Increment

The Precedence of the Arithmetic Operators

Highest	++ – –
	– (unary minus)
	* / %
Lowest	+ –

Relational and Logical Operators

- In C, true is any value other than zero. False is zero.
- The truth table for the *logical operators* is shown here using 1's and 0's.

p	q	p && q	p || q	!p
0	0	0	0	1
0	1	0	1	1
1	1	1	1	0
1	0	0	1	0

Relational and Logical Operators

Relational Operators

Operator	Action
>	Greater than
>=	Greater than or equal
<	Less than
<=	Less than or equal
= =	Equal
!=	Not equal

Logical Operators

Operator	Action
&&	AND
||	OR
!	NOT

Bitwise Operators

- *Bitwise operation* refers to testing, setting, or shifting the actual bits.

Operator	Action
&	AND
|	OR
^	Exclusive OR (XOR)
~	One's complement (NOT)
>>	Shift right
<<	Shift left

Expressions

- An *expression* in C is any valid combination of operators, constants, functions, and variables.
- Most expressions tend to follow the general rules of algebra.

Order of Evaluation

Highest	( ) [ ] ->.	
	! ~ ++ – – – (type) * & sizeof	
	* / %	&
	+ –	^
	<< >>	|
	< <= > >=	&&
	== !=	||
		?:
		= += -= *= /= etc.
	Lowest	,

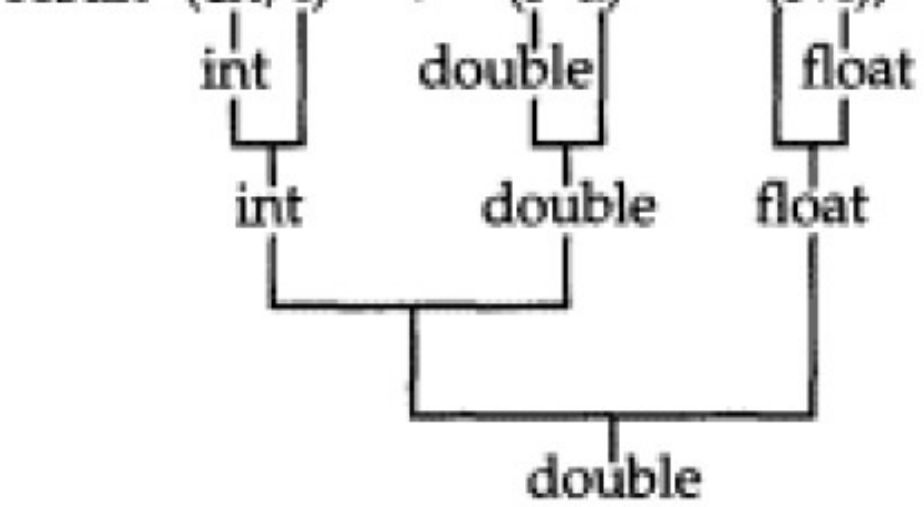
Type Conversion in Expressions

- The compiler converts all operands up to the type of the largest operand, which is called *type promotion*.
- All `char` and `short` int values are automatically elevated to `int`.

IF an operand is a **long double**
THEN the second is converted to **long double**
ELSE IF an operand is a **double**
THEN the second is converted to **double**
ELSE IF an operand is a **float**
THEN the second is converted to **float**
ELSE IF an operand is an **unsigned long**
THEN the second is converted to **unsigned long**
ELSE IF an operand is **long**
THEN the second is converted to **long**
ELSE IF an operand is **unsigned int**
THEN the second is converted to **unsigned int**

A Type Conversion Example

```
char ch;
int i;
float f;
double d;
result=(ch/i)  +  (f*d)  -  (f+i);
```



Casts

- You can force an expression to be of a specific type by using a *cast*.
- The general form of a cast is
  `(type) expression`

```
(float) x/2
```