3. Operators, Control Structures

14 February 2024 08:44

Relational operators

Relational operators do not cascade in C

evaluated left to right

$5 > 4 > 3$

$1 > 3$

0

To overcome this, use logical operators

Logical operators

&& → logical AND

|| → logical OR

! → logical NOT

To use the earlier example:

$(5 > 4)$ && $(4 > 3)$

Short circuit evaluation

Logical operators in C follow the same short circuit evaluation as in Python

Eg:
```
int d = 0;
int e = (d) && (d++);
```
e = 0

d = 0 → why?

&& acts as a sequence point operator; the first expression is always evaluated regardless of operator precedence

Bitwise operators

Works similarly to logical operators but works bit by bit

& → bitwise AND

| → bitwise OR

^ → bitwise XOR

~ → one's complement

<< → left shift

NOTE:

- %u is used for bitwise results so that only magnitude is considered (no sign)
- Negative numbers are stored in their 2's complement form.

~ → one's complement

<< → left shift

>> → right shift

- Negative numbers are stored in their 2's complement form.

~2 (%d) → -3

~2 (%u) → some big no.

Note:

2 << 3 = 2×2^3 = 16

2 >> 3 = $2 \div 2^3$ = 0 (only quotient)

- n << 1 → Multiplying by 2
  n >> 1 → Dividing by 2
- n & 1
  - 0 ⇓ even
  - 1 ⇓ odd
- To set the i^{th} bit to be 1
  n | (1<<i)
- To clear the i^{th} bit in n
  n & ~(1<<i)
- To swap two variables:
  a = a ^ b
  b = a ^ b
  a = a ^ b
- To check if the i^{th} bit in a variable is 1
  n & (1<<i)
  - 0 → not set
  - 1 → set

Ternary operator

Takes 3 arguments

?: → conditional operator

(expr1) ? (expr2) : (expr3)

if true: (expr2) if (expr1) is true

if false: (expr3) if (expr1) is false

Type casting

Say you have two int values but you want the result as a float

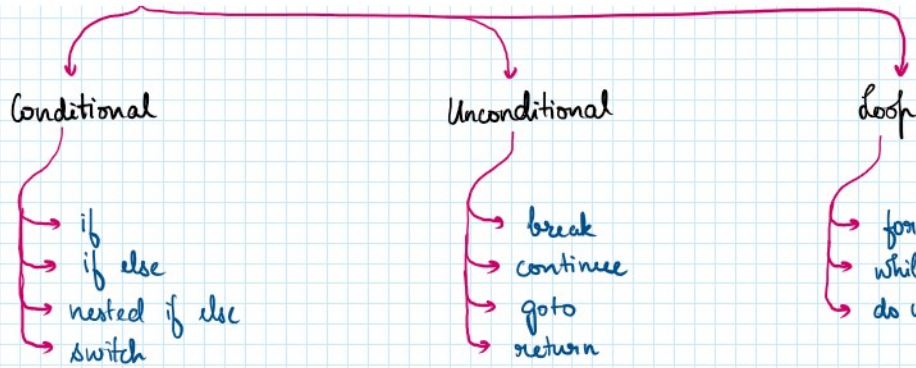
num1 → int
num2 → int

res = (float) num1 / num2

(float) num1 converts num1 value to float

DOES NOT change type of variable

CONTROL STRUCTURES



IF - ELSE

```
if (cond")
{
...
}
```

no semicolon; becomes null statement if semicolon is used

```
if (cond")
{
...
}
else
{
...
}
```

else construct cannot exist without preceding if

```
if (cond")
{
...
}
else if (cond")
{
...
}
else
{
...
}
```

```
if (cond")
{
   if (cond")
   {
   ...
   }
   else
   {
   ...
```

NOTE:

Semicolons are needed inside the body of the if/else conditions

NOTE:

Remembers that C uses symbolic logical operators.

- && → AND
- || → OR
- ! → NOT

NOTE:

Curly braces are not required if the block contains only one statement.

- if-else → one statement
- if, if → two statements

```
    }
}
```

WHILE LOOP → also called top testing loop, entry controlled loop

```
i = 1
while (i<4)        // condition
{
   printf ("%d", i);
   i++;            // loop updation
}
```

FOR LOOP

```
for ( init; cond"; updation)
      (1)   (2)        (4)
{
... (3)
}
```

→ mandatory even if you don't specify updation in loop header

▨ - flow of execution

DO-WHILE LOOP → exit controlled loop

```
do
{
...
} while (cond");
```

→ will be executed at least once

SWITCH STATEMENT

```
switch (integral expression)
{
   case int1 : //block1
               break;
   case int2: // block 2
               break;
   default:   // block n
               break;
}
```

→ {} not required for blocks here

→ takes control outside the switch statements; can also be used in loop

→ not mandatory; executed when integral expression evaluates to some value that does not have a specific case associated with it

NOTE:
You cannot have duplicate cases

Example

```
# include <stdio.h>
```

```
int main ()
{
 char c;
 printf("Enter a character: ");
 scanf("%c", &c);
 switch (c)
 {
  case 'a':
  case 'e':
  case 'i':
  case 'o':
  case 'u':
  case 'A':
  case 'E':
  case 'I':
  case 'O':
  case 'U':
        printf("Vowel \n");
        break;
  default:
        printf("Consonant \n");
        break;
 }
```

BREAK & CONTINUE

- BREAK → takes control outside loop/switch statement
- CONTINUE → takes control to the beginning of the next iteration of a loop

GOTO

- Unconditional statement that takes control to any specified label
- Cannot jump between functions
- Can be used to exit multiple nestings
- Labels used must follow identifier naming rules

Example

```
int i=1, sum = 0, n;
scanf("%d", &n);
```

```
begin: if (i>n)
           goto end;
       sum+=i
       i++;
       goto begin
end: printf("Sum: %d", sum)
```