4 – Operátory a výrazy

Příklad 4.1
Tento program objasňuje použití operátoru modulo (zbytek po dělení). Nejprve promyslete, co program dělá, poté naprogramujte.

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

int main( void )
{
  int c, j = 0;
  printf( "Zadej retezec, ktery zpracujeme: ");
  while ((c = getchar()) != '\n')
  {
    if (j % 7 == 0) /* j delitelne 7 ? */
      printf( "\n" );
    putchar( c );
    j++;
  }
  exit( 0 );
}
```

Příklad 4.2
Pro obecnější chod programu je lépe použít následující funkci, která umožňuje obecné rozdělení textu. Tato funkce může být užita např. při tvorbě textového editoru.

```
#include <stdio.h>
void rozdel_radek( int interval )
{
  int c, j = 1;
  while ((c = getchar()) != '\n')
  {
    putchar( c );
    if (j%interval == 0)
      printf( "\n" );
    j++;
  }
}
```

Příklad 4.3
Vyhodnoťte následující výrazy, pro každý řádek předpokládejte :

```
int j = 0, m =1, n = -1;
float x =2.5, y = 0.0;

- x + j == y > n > m
x + = (y >= n)
++j == m != y * 2
```

Příklad 4.4
Co vytiskne tento program.?

```
#include <stdio.h>
main()
{
  int i = 4, j = 3;

  j += i;       printf("i = %d, j = %d \n", i, j);
```

```
  j /= --i;      printf("i = %d, j = %d \n", i, j);
  j *= i - 2;   printf("i = %d, j = %d \n", i, j);
}
```

Příklad 4.5

Odhadněte, zda je mezi následujícími programy nějaký rozdíl, případně jaký. Poté naprogramujte a ověřte.

a)

```
#include <stdio.h>
#include <stdlib.h>
int main( void )
{
  int j = 5, k = 5;
  printf( "j: %d\t k: %d\n", j++, k );
  printf( "j: %d\t k: %d\n", j, k );
  exit( 0 );
}
```

b)

```
#include <stdio.h>
#include <stdlib.h>
int main( void )
{
  int j = 5, k = 5;
  printf( "j: %d\t k: %d\n", ++j, k );
  printf( "j: %d\t k: %d\n", j, k );
  exit( 0 );
}
```

Příklad 4.6

Co vypíše tento program ?

```
#include <stdio.h>
int main()
{
  int i = 2, j = 1;
  j = j  &&  (i = 2);      printf("j = %d, i = %d \n", j, i);
  i = 1; j = 1;
  j = 2 &&  (i == 2);   printf("j = %d, i = %d \n", j, i);
  i = 1; j = 2;
  j = i ||  (i / 2);           printf("j = %d, i = %d \n", j, i);
  i = 1; j = 1;
  j = !j  &&  (i = i + 2);   printf("j = %d, i = %d \n", j, i);
}
```

Příklad 4.7

Co vypíše tento program ?

```
#include <stdio.h>
main()
{
 int i, k, j =5;
i = (j == 2) ? 1 : 7;  printf("i = %d \n", i);
k = (i >j) ? 2 : j;  printf("k = %d \n", k);
}
```

Příklad 4.8

Pro každý výraz předpokládejte následující vstupní hodnoty a výraz vyhodnoťee:

```
int m = 6, n = 5;
float x = 3.5, y = 1.0;
```

```
m + n + x + y
m + x * n + y
x / y + m /n
x - y * m + y / n
x / 0
```

Příklad 4.9

Pro každý výraz předpokládejte následující vstupní hodnoty a výraz vyhodnoťte:

```
int m = 3, n = 4;
float x = 3.5, y = 1.0;

m + = n + x - y
m / = x * n + y
n %  = n + m
x +  = y -  = m
```

Příklad 4.10

Vyhodnocení následujícího výrazu není dáno normou jazyka C, je implementačně závislé. Zjistěte, jak jazyk C, s kterým pracujete, výraz vyhodnotí. Za podmínky, že j = 4, jaké hodnoty nabývá výraz j * j ++.

Příklad 4.11

Pro každý výraz předpokládejte následující vstupní hodnoty a výraz vyhodnoťte:

```
int m = 3, n = -2, j = 0;

m++ - --j
m += ++j *3
m++ * m++ ( implementačně závislé )
```

Příklad 4.12

V následujícím výrazu je užit operátor čárky. Zamyslete se nad tím, jaké hodnoty nabývají po proběhnutí celého cyklu j a k. Poté ověřte programem. Dále zkuste zaměnit cyklus for cyklem while se stejným významem.

```
for ( j = 0, k = 100; k- j > 0; j++, k--);
```

Příklad 4.13

Co vypíše tento program ?

```
#include<stdio.h>
main()
{
int r1 = 15, m = 121, t =15;
int x = 2,y = 3, i;         /*  tento radek zkuste zamenit int x = 2, y = 2, i  */
i= r1> m || r1 == t;  printf("%d,",i);
i = !( x == y) && !y || x== y && y;
printf ("%d ",i);
}
```

Příklad 4.14

Co vypíše tento program ?

```
#include<stdio.h>
main()
{
int i, z =112;
i = z >= 60 && z <= 100;
printf("%d,",i);
}
```

Příklad 4.15

Co vypíše tento program ?

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

```
main()
{
int a,b = 3;
a = b++;     printf("%d,%d\n", a, b );
a = --b + 3; printf("%d,%d\n", a, b );
a = -b ++;   printf("%d,%d\n", a, b );
a = ++b;     printf("%d,%d\n", a, b );
b++;         printf("%d,%d\n", a, b );
}
```

Příklad 4.16
Co vypíše tento program ?

```
#include<stdio.h>
int i, a = 6, b=7;
double y, x =7.0;
main()
{
i = a*b;        printf("%d\n",i );
y = a*x;        printf("%f\n", y);
i = 5/3;        printf("%d\n", i );
i = -5/3;       printf("%d\n",i );
i = 11%3;       printf("%d\n",i );
i =11% -3;      printf("%d\n",i );
i = -11%3;      printf("%d\n",i );
i = -11% -3;    printf("%d\n",i );
i = 100/11*6 %12;  printf("%d\n",i );
}
```

Příklad 4.17
Rozhodněte zda výraz y = 1 + x <<2 se vyhodnotí jako y = (1 + x) <<2 nebo y = 1 + (x <<2)

Příklad 4.18
Pro každý výraz předpokládejte a poté vyhodnoťte:
int m = 6, n = 5, j = 1;
float x = 3.5, y = 1.0;

```
j > m
m / n < x
j < = m > = n
j < = x  = = m
-x + j  = = y > n > m
x + = (y > = n)
++j = = m ! = y * 2
```

Příklad 4.19
Rozhodněte, zda následující dva fragmenty programu jsou totožné:

```
1. if ( (a < b) && (b<c))
          p1;

2. if (a<b)
       if (b<c)
          p1;
```

Příklad 4.20
Rozhodněte, zda následující tři fragmenty programů jsou všechny totožné nebo pouze některé nebo žádné:

```
1. if ( (a < b) && (b<c))
```

```
        p1;
  else
        p2;

2. if (a<b)
    if (b<c)
        p1;
    else
        p2;

3. if (a < b)
    if (b < c)
        p1;
    else
        p2;
    else
        p2;
```

Příklad 4.21
Pro každý výraz předpokládejte a poté vyhodnoťte:
int m = 1, n = -1, j = 0;
float x = 2.5, y = 0.0;

```
j && m
j < m && n <m
m + n || !j
x * 5 && 5 || m / n
j <= 10 && x >= 1 && m
!x || !n || m + n
x * y < j + m ||n
(x > y) + !j || n++
(j ||m) + (x || ++n)
```

Příklad 4.22
Za předpokladu :
int a = 1, b = 2 , c = 5, d = 4;
vyhodnoťte následující výrazy a určete velikost proměnné c. Nazapomeňte na to, že céčko provádí tzv. zkrácené vyhodnocování.

```
if ((a<b) && (c== d)
if ((a<b) && (c== d++)
```

Příklad 4.23
Vyhodnoťte výsledek působení operátoru posunu:
3 << 1
255 >>2
8 << 8
1<< 15

Příklad 4.24
Vyhodnoťte výsledek působení operátoru posunu (pozor, u záporných čísel je výsledek závislý na implementaci):
-5 >>2

Příklad 4.25
Vyhodnoťte následující výrazy.
9430 & 5722
9430 | 5722
9430 ^ 5722
~9430

Příklad 4.26
Proveďte u následujících výrazů uzávorkování stejně, jako provede kompilátor. Poté naprogramujte a ověřte .
a) a = b *c == 2;
b) a = f(x) && a > 100;
c) a == b && x ! = y;
d) a = b += 2 +f(x);

Příklad 4.27
Co vypíše tento program ?
```
#include <stdio.h>
main()
{
  int i, j;
  float f;
  i = 5, j = 2;                /* pouziti operatoru carky */
  f = (float) i / j;    printf("%f\n", f);
  f = i / (float) j;    printf("%f\n", f);
  f = (float) i / (float) j;     printf("%f\n", f);
  f = (float) (i / j);    printf("%f\n", f);
  f = i / j;    printf("%f\n", f);
}
```

Příklad 4.28
Co vypíše tento program ?
```
#include <stdio.h>
main()
{
  int i, j;
  printf("Zadej dve cela cisla : ");
  scanf("%d %d", &i, &j);
  printf(" %d \n", (i < j) ? i : j);
  printf(" %d \n", (i > j) ? i : j);
}
```

Příklad 4.29
Co vypíše tento program ?
```
#include <stdio.h>
main()
{
  int i;   i = 5;
  printf("%d \n", i == 8);
  printf("%d \n", i = 8);
  printf("%d \n", i == 8);
}
```

Příklad 4.30
Vytvořte program, který se bude hodit na vyhodnocování testů. Test umožňuje pouze odpovědi ano-ne. Za předpokladu, že typ long int je dlouhý 4B tj. 32 bitů, můžeme do jedné proměnné tohoto typu uložit celkem 32 informací (odpovědí) ano-ne (nebo jiných informací, které nabývají pouze 2 hodnoty: zapnuto - vypnuto, černý - bílý, muž - žena atd.). Předpokládejme, že máme uložit výsledky testů velkého množství studentů; potom tento způsob ukládání ušetří mnoho místa.

a) První operací, kterou musí program zvládnout, je uložení 32 ano (1) -ne (2).

Na to použijte cyklu, ve kterém se budete uživatele 32 krát ptát na ano (‘a’ či ‘A’) nebo ne (‘n’ či ‘N’). Za předpokladu, že odpovědí je 32, užijte následující segment programu:
```
.......
long odpovedi = 0;
```

```
for  j = 0; j <= 31; j++)
{
  scanf(" %c", &c);          /* načítam odpoved */
  if (c == 'a' || c == 'A')      /* kdy je ano, jdu dal */
  odpovedi |= 1L << j;          /* provadim zmenu promenne odpovedi */
}
.....
```

Zde je klíčovou částí následující řádek :

odpoved |= 1L << j;

Zde jednak dochází k následujícímu posunu 1 v long int :

j	1<<j
0	00000000 00000000 00000000 00000001
1	00000000 00000000 00000000 00000010
2	00000000 00000000 00000000 00000100
3	00000000 00000000 00000000 00001000

atd.

Výsledná hodnota získaná posunem po každém průchodu cyklu je poté operací bitový OR 'sloučena' s dosavadním výsledkem (odpovedi).
Proměnná odpoved typu long int je řadou 32 nul a jedniček. Nutno si uvědomit, že pokud první odpověď byla ano, je bit zcela vpravo jednička. Pokud poslední odpověď byla ano, je bit zcela vlevo jednička.

b) Jako další část programu vytvořte porovnání se správnými hodnotami v testu. Nejlépe pomocí XOR po bitech. Nezapoměňte na již zmíněné opačné pořadí. Správné hodnoty možno zadat např. jako šestnáctkové long int číslo pomocí příkazu preprocesoru #define . To umožní snadnou změnu programu při jiné verzi test. nebo je možno správnou hodnotu jednoduše načíst.

c) Jako poslední část programu, vyhodnoťte procentuální úspěšnost studenta.

Dále by bylo možno celý program upravit tak, aby se v cyklu ptal na výsledky jednotlivých studentů, příp. aby na konci výsldky srovnal podle pořadí atd.

Příklad 4.31
Použitím operátoru posunu určete velikost největšího čísla int, které vaše C umí.

Příklad 4.32
Napište funkci, která má jako parametr číslo int a vypíše ho ve dvojkové reprezentaci. Aby funkce byla co nejobecnější, pro velikost čísla int určete operátor sizeof().

Příklad 4.33
Napište funkci, která čte číslo ve dvojkové podobě a konvertuje ho na hexadecimální reprezentaci.

Od RNDr. Dagmar Brechlerové.