Imperative programming

3. Expressions, operators, expression evaluation

Zoltán Porkoláb

Expressions

- Goal of the first languages: to write mathematical expressions (FORTRAN)
- Am expression has a type and a value
- For languages with static type system the type is decided in compile time
- The value is decided usually in run-time, but sometimes also in compile time
- The meaning of an expression is decided by
  - Parentheses
  - Operator **precedence**
  - **Associativity** rules
- Some operators have **side effect**

```
int f(int i, int a, int b, int c, int d)
{
  ++i;                  // i+1, side effect: i = i + 1
  i++;                  // i,   side effect: i = i + 1
  return i+a*b/c*d;     // i + ( ( ( a * b ) / c ) * d )
}
```

Operators

group	operators	associativity
Postfix	++ -- ( ) [ ] . -> (type){list}C11	Left-to-right
Unary	++ -- + – ! ~ (type) * & sizeof _AlignofC11	Right-to-left
Multiplicative	* / %	Left-to-right
Additive	+ –	Left-to-right
Bit-shift	<< >>	Left-to-right
Relational	< <= > >=	Left-to-right
Equality	== !=	Left-to-right
Bit-wise and	&	Left-to-right
Bit-wise excl. and	^	Left-to-right
Bit-wise or	|	Left-to-right
Logical and	&&	Left-to-right
Logical or	||	Left-to-right
Ternary conditional	? :	Right-to-left
Assignment	= *= /= %= += –= <<= >>= &= ^= |=	Right-to-left
Comma	,	Left-to-rigth

Expressions

```
int f(int t[])
{
  int j = 42;
  int k = j++;  // k == 42, j == 43
  int n = ++k;  // n == 43, k == 43

  int i = 0;
  while ( i < 10 )   // suppose t[10] is declared previously
  {
    --t[i++]; // t[i] = t[i]-1; i = i+1
  }

  for ( int i = 0, i < 10; ++i)   // better
  {
    --t[i];   // t[i] = t[i]-1
  }
}
```

Expressions

```
struct Student
{
  char neptun_code[6];
  char name[20];
  int  semester;
};

int f(char *name)
{
  struct Student aPoorStudent;

  aPoorStudent.semester = 1;
  strncpy(aPoorStudent.name, name, 19);  // name[0] ... name[18]
  aPoorStudent.name[19] = '\0';          // defensive safety

  struct Student *ptr = &aPoorStudent;  // pointer ptr points
                                        // to aPoorStudent
  ++(*ptr).semester;    //   ++aPoorStudent.semester
  ++ptr->semester;      //   ++(*ptr).semester
}
```

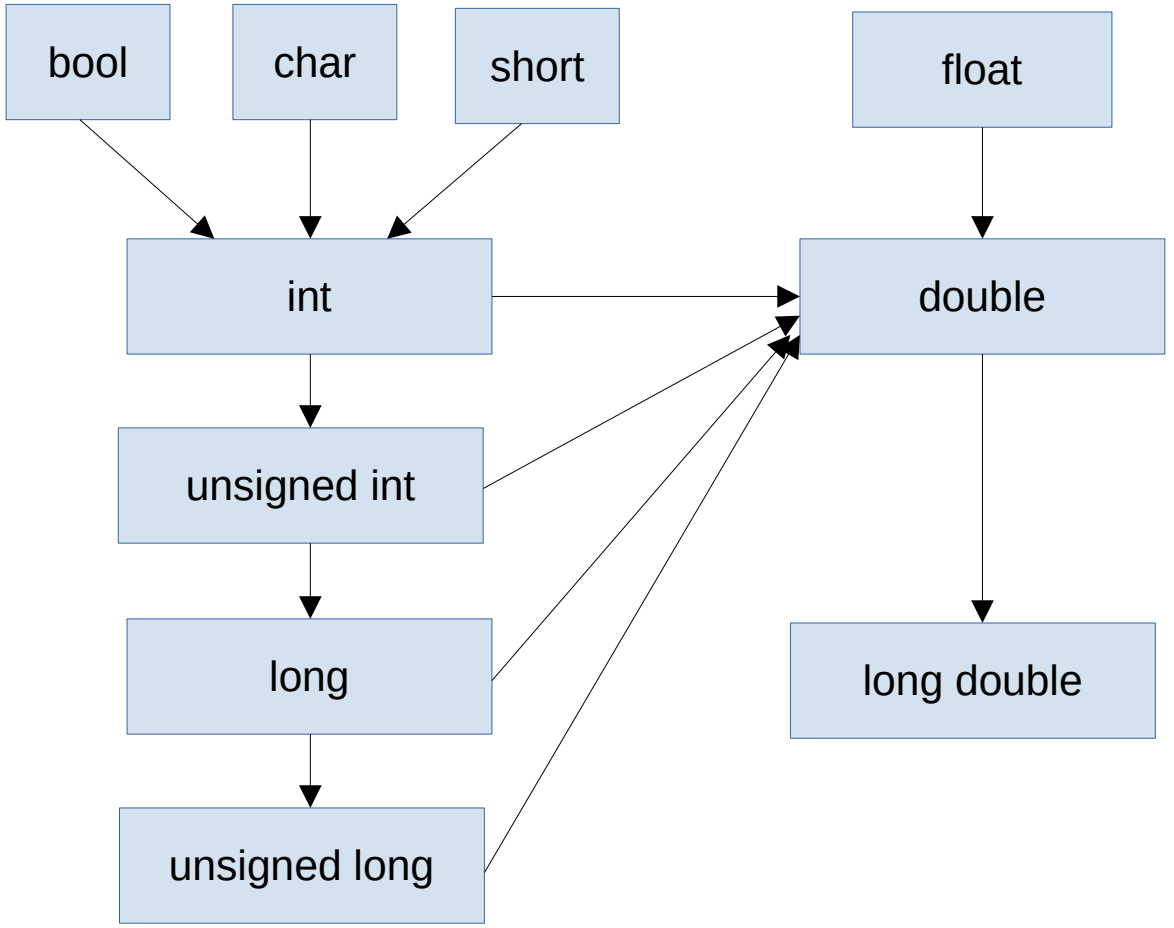
Expressions

```
int f(char *ptr, struct Student *sp)
{
   unsigned int x = ~(~0u >> 1);   // 100000000...0000000000
   int y = !++*++ptr;

   if ( y % 2 )  {
      printf("y is odd\n");
   } else {
      printf("y is even\n");
   }

   printf("y is %s\n", y % 2 ? "odd" : "even");

   if ( sp != NULL && sp->semester > 1 )  // NULL is a macro
   {
     printf("%s", sp->name);
   }

   if ( sp && sp->semester > 1 )     // sp != NULL
   {
     printf("%s", sp->name);
   }
}
```

Assignments

```
struct Point
{
  double x;
  double y;
};

int f(int y, struct Point *ap, struct Point *bp)
{
   double pi = 3.14;
   int x = y + 3;
   x = x + y;
   x += y;

   t[f(a,b)] = t[f(a,b)] + 1;
   t[f(a,b)] += 1;
   ++t[f(a,b)];

   *ap = *bp;  // ap->x = bp->x; ap->y = bp->y;

   x = y = 5;                      // y = 5; x = y;
   double pi2 = x = pi = 3.14;  // pi2 == 3 !!

}
```

Implicit conversions



Expression evaluation

- Some operators are **unevaluated** (sizeof, _Alignof).
- No run-time action.

```
int main()
{
  size_t int_size = sizeof(printf("%d",42)); // no output
  printf("size of int: %u\n", int_size);     // e.g. 4
  return 0;
}
$ gcc -Wall -Wextra plusplus.c
4
$
```

Expression evaluation

- What will print the following program?

```
#include <stdio.h>

int main()
{
  int i = 1;
  printf("i == %d, ++i == %d\n", i, ++i);
  return 0;
}
```

```
$ gcc plusplus.c
$ ./a.out
i == 1, ++i == 2
```

Expression evaluation

- What will print the following program?

```
#include <stdio.h>

int main()
{
  int i = 1;
  printf("i == %d, ++i == %d\n", i, ++i);
  return 0;
}
```

```
$ gcc plusplus.c
$ ./a.out
i == 2, ++i == 2
```

Expression evaluation

- What will print the following program?

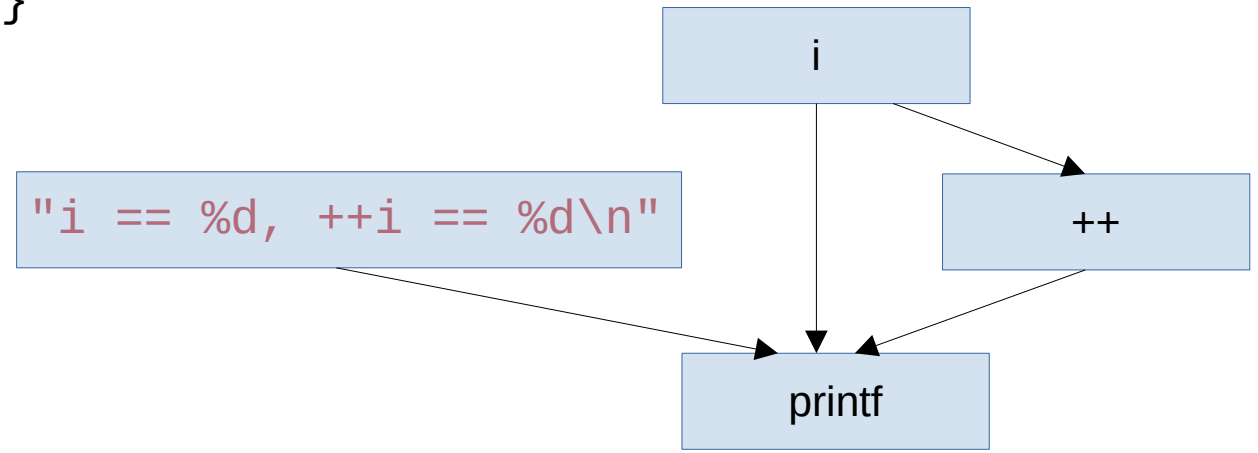
```
#include <stdio.h>

int main()
{
  int i = 1;
  printf("i == %d, ++i == %d\n", i, ++i);
  return 0;
}
```

```
$ gcc -Wall -Wextra plusplus.c
plusplus.c: In function ‘main’:
plusplus.c:6:38: warning: operation on ‘i’ may be undefined [-
Wsequence-point]
 6 |   printf( "i == %d, ++i == %d\n", i, ++i);
   |                                      ^~~
```

Expression evaluation

- What will print the following program?

```c
#include <stdio.h>

int main()
{
  int i = 1;
  printf("i == %d, ++i == %d\n", i, ++i);
  return 0;
}
```



Expression evaluation

- The meaning of an expression is decided by
  - Parentheses
  - Operator **precedence**
  - **Associativity** rules
- Some operators have **side effect** a = b++
- The **order of the evaluation** is usually not defined
- Except:
  - Shortcut logical operators: a && b and a || b
  - Evaluation of condition of ternary conditional a ? b : c
  - Comma operator a , b
  - All the function parameters are evaluated before the function body (but we do not now the order of the parameter evaluation)

Expression evaluation

- What will print the following program?

```c
#include <stdio.h>

int main()
{
  int i = 1;
  printf("i == %d", i);      // sequence point
  printf(", ++i == %d\n", ++i);
  return 0;
}
```

```
$ gcc -Wall -Wextra plusplus.c
$ ./a.out
i == 1, ++i == 2
```

Expression evaluation

- What will print the following program?

```c
#include <stdio.h>

int main()
{
  int i = 1;
  printf("i == %d", i);       // sequence point
  ++i;                        // sequence point
  printf(", ++i == %d\n", i);
  return 0;
}
```

```
$ gcc -Wall -Wextra plusplus.c
$ ./a.out
i == 1, ++i == 2
```

Expression evaluation

- What will print the following program?

```
#include <stdio.h>
int main()
{
    int t[10];
    int i = 0;
    while( i < 10 )
    {
        t[i] = i++;

    }

    for ( i = 0; i < 10; ++i )
    {
        printf("%d ", t[i]);
    }
    return 0;
}
$ gcc loop.c
$
```

Expression evaluation

- What will print the following program?

```
#include <stdio.h>
int main()
{
    int t[10];
    int i = 0;
    while( i < 10 )
    {
        t[i] = i++;   // unefined behavior

    }

    for ( i = 0; i < 10; ++i )
    {
        printf("%d ", t[i]);
    }
    return 0;
}
```

```
$ gcc loop.c
$ ./a.out
613478496 0 1 2 3 4 5 6 7 8
```

Expression evaluation

- What will print the following program?

```
#include <stdio.h>
int main()
{
    int t[10];
    int i = 0;
    while( i < 10 )
    {
        t[i] = i;   // sequence point
        ++i;
    }

    for ( i = 0; i < 10; ++i )
    {
        printf("%d ", t[i]);
    }
    return 0;
}
$ gcc loop.c
$ ./a.out
0 1 2 3 4 5 6 7 8 9
```

Expression evaluation

- What will print the following program?

```
#include <stdio.h>
int main()
{
    int t[10];
    for ( i = 0; i < 10; ++i )
    {
        t[i] = i;
    }

    for ( i = 0; i < 10; ++i )
    {
        printf("%d ", t[i]);
    }
    return 0;
}
```

```
$ gcc loop.c
$ ./a.out
0 1 2 3 4 5 6 7 8 9
```

Expression evaluation

```
#include <stdio.h>

int f() {   printf("f\n");  return 2;  }
int g() {   printf("g\n");  return 1;  }
int h() {   printf("h\n");  return 0;  }

void  func() { printf("(f() == g() == h()) == %d\n",
                                        f() == g() == h()); }

int main()
{
    func();
    return 0;
}

$ ./a.out
f
g
h
(f() == g() == h()) == 1
```

Expression evaluation

```
#include <stdio.h>

int f() {   printf("f\n");  return 2;  }
int g() {   printf("g\n");  return 1;  }
int h() {   printf("h\n");  return 0;  }

void  func() { printf("(f() == g() == h()) == %d\n",
                                        f() == g() == h()); }

int main()
{
    func();
    return 0;
}
```

```
( f()  ==  g() ) ==  h()
  2    ==   1
       0    ==   0
            1
```

```
$ ./a.out
f
g
h
(f() == g() == h()) == 1
```

Expression evaluation

```
#include <stdio.h>

int f() {   printf("f\n");  return 2;  }
int g() {   printf("g\n");  return 1;  }
int h() {   printf("h\n");  return 0;  }

void  func() { printf("(f() == g() == h()) == %d\n",
                                        f() == g() == h()); }

int main()
{
    func();
    return 0;
}

$ ./a.out
h
g
f
(f() == g() == h()) == 1
```

Traps and pitfalls

```c
#include <stdio.h>

int main()
{
    int mask = 0x01;
    if ( mask & 0x10 == 16 )
    {
        printf("This is strange!\n");
    }
    return 0;
}
```

```
$ gcc mask.c
$ ./a.out
This is strange!
```

Traps and pitfalls

```c
#include <stdio.h>

int main()
{
    int mask = 0x01;
    if ( mask & 0x10 == 16 )  // mask & (0x10 == 16 )
    {
        printf("This is strange!\n");
    }
    return 0;
}
```

```
$ gcc mask.c
$ ./a.out
This is strange!
```

```
mask = 1
0x10 = 16
mask & 0x10 == 0
0x10 == 16  == 1
```

Traps and pitfalls

```c
#include <stdio.h>

int main()
{
    int mask = 0x01;
    if ( (mask & 0x10) == 16 )  // (mask & 0x10) == 16
    {
        printf("This is strange!\n");
    }
    return 0;
}
```

```
$ gcc mask.c
$ ./a.out
This is strange!
```

```
mask = 1
0x10 = 16
mask & 0x10 == 1
1 == 16     == 0
```

Traps and pitfalls

```c
#include <stdio.h>

int main()
{
    int i = 0;
    if ( i = 1 )
    {
        printf("This is strange!\n");
    }
    return 0;
}
```

```
$ gcc one.c
This is strange!
```

Traps and pitfalls

```
#include <stdio.h>

int main()
{
    int i = 0;
    if ( i == 1 )   // do not mistake = and ==
    {
        printf("This is strange!\n");
    }
    return 0;
}
```

```
$ gcc one.c
$
```

Traps and pitfalls

```c
#include <stdio.h>

int main()
{
    int i = 0;
    if ( 1 = i )
    {
        printf("This is strange!\n");
    }
    return 0;
}
```

```
$ gcc one.c
one.c: In function ‘main’:
one.c:5:12: error: lvalue required as left operand of
assignment
    5 |     if ( 1 = i )
      |            ^
```

Traps and pitfalls

```c
#include <stdio.h>

int main()
{
    int i = 0;
    if ( 1 = i )    // Yoda condition
    {
        printf("This is strange!\n");
    }
    return 0;
}
```

```
$ gcc one.c
one.c: In function ‘main’:
one.c:5:12: error: lvalue required as left operand of
assignment
    5 |     if ( 1 = i )
      |            ^
```