Высшая школа экономики
Факультет компьютерных наук
Департамент программной инженерии

Алгоритмы и алгоритмические языки
Лекция 4

30 сентября 2025 г.

Операции над целочисленными данными

Арифметические

одноместные: изменение знака (`-`), одноместный плюс (`+`)
двухместные: сложение (`+`), вычитание (`–`), умножение (`*`), деление нацело (`/`), остаток от деления нацело (`%`)
$(a/b) \times b + (a\%b) == a$

Отношения (результат 0/1 типа `int`)

больше (`>`), больше или равно (`>=`), меньше (`<`), меньше или равно (`<=`)

Сравнения (результат 0/1 типа `int`)

равно (`==`), не равно (`!=`)

Логические

отрицание (`!`), конъюнкция (`&&`), дизъюнкция (`||`)
ложное значение — 0, истинное — любое ненулевое
«ленивое» вычисление `&&` и `||`

Операции присваивания

Побочные эффекты: изменение объекта, вызов функции

`lvalue = rvalue`

- `lvalue` — выражение, указывающее на объект памяти
- `rvalue` — выражение, генерирующее значение
  Пример: `a = b = c = d = 0;`

Укороченное присваивание: `lvalue op= rvalue`, где `op` — двухместная операция
Пример: `a += 15;`

Инкремент и декремент: `++` и `--`
префиксные и постфиксные

Последовательное вычисление: операция запятая `,`
Пример: `a = (b = 5, b + 2);`

Точки следования

Побочные эффекты: изменение объекта, вызов функции

Точка следования (sequence point): момент во время выполнения программы, в котором все побочные эффекты предыдущих вычислений закончены, а новых — не начаты

- первый операнд `&&`, `||`, `,`
- окончание *полного* выражения (full expression)
- между вычислением фактических параметров и вызовом функции

Между двумя точками следования изменение значения переменной возможно не более одного раза[1].

```
(a=2) + (a=3)
i++ + ++i
```

[1] В последних стандартах терминология несколько иная (sequenced before, unsequenced, indeterminately sequenced): точка следования влечёт частичный порядок, его отсутствие делает возможным любые варианты.

Форматный ввод-вывод

```
#include <stdio.h>
int main (void) {
  int s = 0;
  int a, b;
  scanf ("%d%d", &a, &b);
  s += a + b;
  printf ("Sum␣is␣%d\n", s);
  return 0;
}
```

Спецификаторы ввода-вывода

спецификатор	печатает/считывает
`%d`, `%ld`, `%lld`	число `int`, `long`, `long long`
`%u`, `%lu`, `%llu`	число `unsigned`, `unsigned long`, `unsigned long long`
`%f`, `%Lf`	печатает `double`, `long double`
`%f`, `%lf`, `%Lf`	считывает `float`, `double`, `long double`
`%c`	символ (`char`)

`%4d`: вывести число типа `int` минимум в четыре символа

`%.5f`: вывести число типа `double` с пятью знаками

`%%`: напечатать знак процента

C23: а теперь и `%b` появился

Функция `scanf` возвращает количество удачно считанных элементов

Пример Си-программы

```
/* Solving a quadratic equation */
#include <stdio.h>
#include <math.h>
int main (void) {
  int a, b, c, d;
  /* Input coefficients */
  if (scanf ("%d%d%d", &a, &b, &c) != 3) {
    printf ("Need␣to␣input␣three␣coefficients!\n");
    return 1;
  }
  if (!a) {
    printf ("That's␣not␣quadratic!\n");
    return 1;
  }
 <...>
```

Пример Си-программы

```
<...>
  d = b*b - 4*a*c;
  if (d < 0)
    printf ("No solutions\n");
  else if (d == 0) {
    double db = -b;
    printf ("Solution: %.4f\n", db/(2*a));
  } else {
    double db = -b;
    double dd = sqrt (d);
    printf ("Solution 1: %.4f, solution 2: %.4f\n",
            (db+dd)/(2*a), (db-dd)/(2*a));
  }
  return 0;
}
```

Преобразование типов

При присваивании `a = b`:

- «Широкий» целочисленный тип в «узкий»: отсекаются старшие биты
- Знаковый тип в беззнаковый: знаковый бит «становится» значащим
  `signed char c = -1; /* sizeof(c) == 1 */`
  `((unsigned char) c)` $\rightarrow$ `255`
- Плавающий тип в целочисленный: отбрасывается дробная часть
- «Широкий» плавающий тип в «узкий»: округление или усечение числа

Явное приведение типов: `(type) expression`

```
d = ((double) a+b)/2;
```

Приведение типов

Неявное приведение типов: происходит, когда операнды двухместной операции имеют разные типы (6.3.1.8)

- Если один из операндов — `long double`, то и второй преобразуется к `long double` (так же для `double` и `float`)
  `long double + double` → `long double + long double`
  `int + double` → `double + double`
  `float + short` → `float + int` → `float + float`
- Если все значения операнда могут быть представлены в `int`, то операнд преобразуется к `int`, так же и для `unsigned int`
  (англоязычный термин — integer promotion)
  `unsigned short(2) + char(1)` → `int(4) + int(4)`
  `unsigned short(4) + char(1)` → `unsigned int(4) + int(4)`
- Если оба операнда — соответственно знаковых или беззнаковых целых типов, то операнд более «узкого» типа преобразуется к операнду более «широкого» типа
  `int + long` → `long + long`
  `unsigned long long + unsigned` →
  `unsigned long long + unsigned long long`

Приведение типов

- Если операнд беззнакового типа более или так же «широк», чем операнд знакового «узкого» типа, то операнд «узкого» типа преобразуется к операнду «широкого» типа
  `int + unsigned long` → `unsigned long + unsigned long`
  `int(4) / unsigned int(4)` → `unsigned int(4) / unsigned int(4)`
  `/* Неверные значения */`
- Если тип операнда знакового типа может представить все значения типа операнда беззнакового типа, то операнд беззнакового типа преобразуется к операнду знакового типа
  `unsigned int(4) + long(8)` → `long(8) + long(8)`
  `unsigned short + long long` → `long long + long long`
- Оба операнда преобразуются к беззнаковому типу, соответствующему типу операнда знакового типа
  `unsigned int(4)+ long(4)` →
  `unsigned long(4) + unsigned long(4)`
- Числа типа `float` не преобразуются автоматически к `double`

Поразрядные операции

`&` (поразрядное И)

`|` (поразрядное включающее ИЛИ)

`^` (поразрядное исключающее ИЛИ)

`<<` (сдвиг влево, нюансы на след. слайде)

`>>` (сдвиг вправо)

- Беззнаковое число — заполнение нулями.
- Знаковое число — заполнение значением знакового разряда (арифметический сдвиг) или нулями (логический сдвиг).

`~` (дополнение до 1, или инверсия)

https://web.archive.org/web/20190915025154/http://www.hackersdelight.org/

```
x & 1          x | 1          x | (1 << 5)          x & (x - 1)
x ^= y, y ^= x, x ^= y          ~x + 1             x | (x + 1)
```

C23: `stdbit.h` и `stdc_...`-функции

`count_ones`, `bit_floor`, `first_trailing_one`...

Целочисленное переполнение

Знаковое переполнение является неопределённым поведением[2] (см. также сдвиги влево <<).

Сдвиги влево (`E1 << E2`): If `E1` has a signed type and nonnegative value, and $\texttt{E1} \times 2^{\texttt{E2}}$ is representable in the result type, then that is the resulting value; otherwise, the behavior is undefined.

Беззнаковое переполнение определено — пропадают старшие биты[3] (см. также сдвиги влево <<).

---

[2] 6.5 5) If an *exceptional condition* occurs during the evaluation of an expression (that is, if the result is not mathematically defined or not in the range of representable values for its type), the behavior is undefined.

[3] 6.2.5 9): A computation involving unsigned operands can never overflow, because a result that cannot be represented by the resulting unsigned integer type is reduced modulo the number that is one greater than the largest value that can be represented by the resulting type.

Операторы

Выражение-оператор: `expression;`

Составной оператор: `{}`

Условный оператор: `if (expr) stmt; else stmt;`

`else` всегда относится к ближайшему `if`:

```
if (x > 2)
    if (y > z)
        y = z;
    else
        z = y;
```

```
if (x > 2) {
    if (y > z)
        y = z;
}
else
    z = y;
```

Операторы

Оператор выбора:

```
switch (expr) {
    case const-expr: stmt;
    case const-expr: stmt;
    default: stmt;
}
```

Оператор `break` — немедленный выход из `switch`.

Операторы

Цикл `while`: `while (expression) stmt;`

Цикл `do-while`: `do { stmt; } while (expression);`

Проверка условия выхода из цикла после выполнения тела

Цикл `for`:

```
for (decl1; expr2; expr3)
    stmt;
```

```
decl1;
while (expr2) {
    stmt;
    expr3;
}
```

`for ( ; ; ) stmt;` — бесконечный цикл

Операторы

Операторы `break` и `continue`: выход из внутреннего цикла и переход на следующую итерацию

Оператор `goto`: переход по метке

```
goto label;
...
label:
```

Областью видимости метки является *вся функция*

Программа: количество дней между двумя датами

```
int main (void)
{
  while (1) {
    int m1, d1, y1, m2, d2, y2;
    int t1, t2;
    int days1, days2, total;

    if (scanf ("%d%d%d%d%d%d", &d1, &m1, &y1,
               &d2, &m2, &y2) != 6)
      break;
    t1 = check_date (d1, m1, y1);
    if (t1 == 1 || (t2 = check_date (d2, m2, y2)) == 1)
      break;
    else if (t1 == 2 || t2 == 2)
      continue;
```

Программа: количество дней между двумя датами

```
    <...>
    days1 = days_from_jan1 (d1, m1, y1);
    days2 = days_from_jan1 (d2, m2, y2);
    total = days_between_years (y1, y2)
            + (days2 - days1);
    printf ("Days␣between␣dates:␣%d,"
            "weeks␣between␣days:␣%d\n",
            total, total / 7);
  }
  return 0;
}
```

Программа: количество дней между двумя датами

```
#include <stdio.h>

static int check_date (int d, int m, int y)
{
  if (!d || !m || !y)
    return 1;
  if (d < 0 || m < 0 || y < 0)
    {
      printf ("%d␣%d␣%d:␣wrong␣date\n", d, m, y);
      return 2;
    }
  return 0;
}
```

Программа: количество дней между двумя датами

```
static int leap_year (int y) {
  return (y % 400 == 0) || (y % 4 == 0 && y % 100 != 0);
}

static int days_in_year (int y) {
  return leap_year (y) ? 366 : 365;
}

static int days_between_years (int y1, int y2) {
  int i;
  int days = 0;

  for (i = y1; i < y2; i++)
    days += days_in_year (i);
  return days;
}
```

Программа: количество дней между двумя датами

```
static int days_from_jan1 (int d, int m, int y) {
  int days = 0;
  switch (m) {

      case 12: days += 30;
      case 11: days += 31;
      case 10: days += 30;
      case 9: days += 31;
      case 8: days += 31;
      case 7: days += 30;
      case 6: days += 31;
      case 5: days += 30;
      case 4: days += 31;
      case 3: days += leap_year (y) ? 29 : 28;
      case 2: days += 31;
      case 1: break;

    }
    return days + d;
  }
```