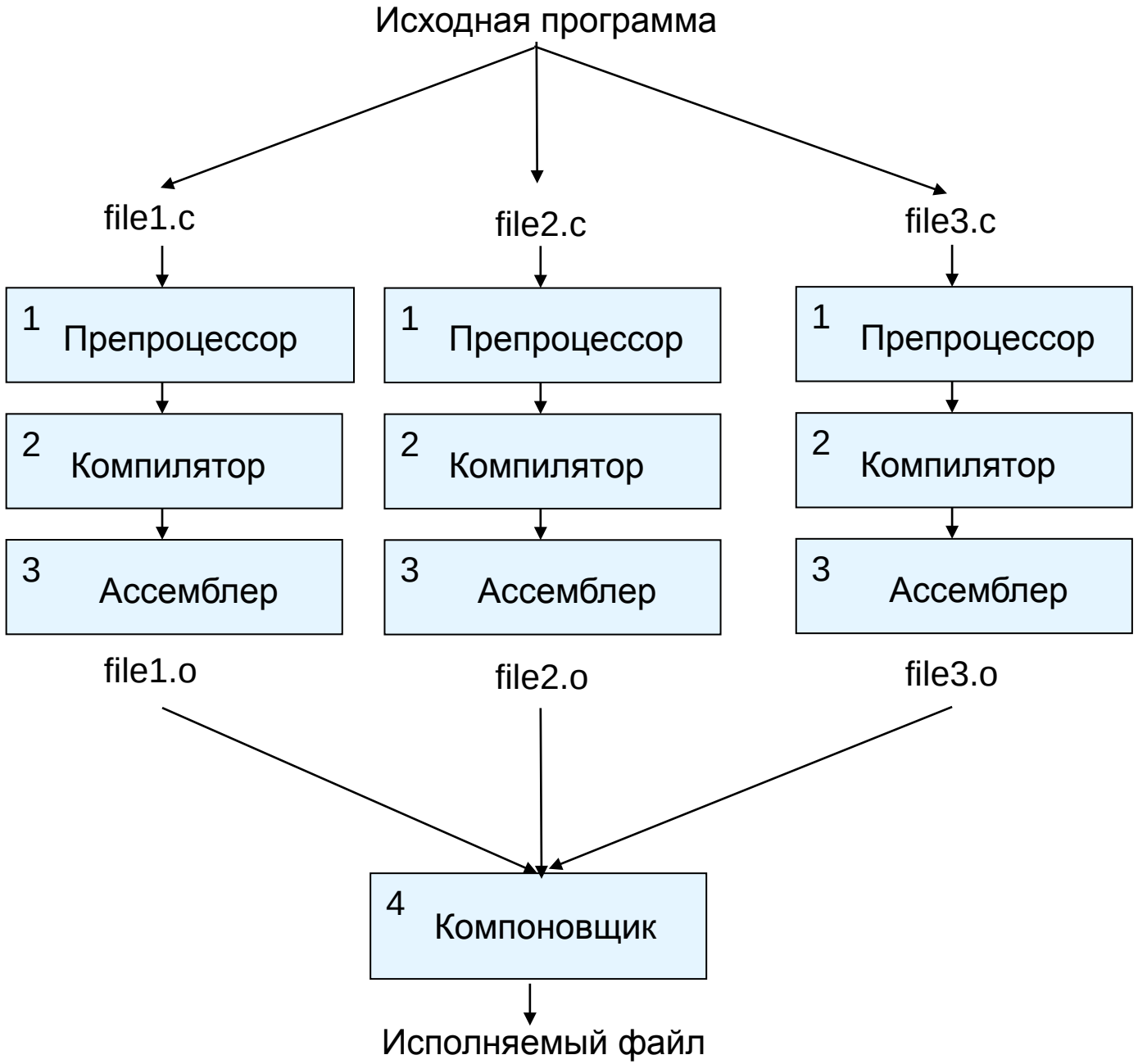
Курс «Алгоритмы и алгоритмические языки»

Лекция 13

Схема компиляции



Препроцессор

- Перед компиляцией выполняется этап препроцессирования. Это обработка программного модуля для получения его окончательного текста, который отдается компилятору.
- Управление препроцессированием выполняется с помощью *директив* препроцессора:

```
#include <...> - системные библиотеки
#include "..." – пользовательские файлы

#define name (parameters) text
#undef name

#define MAX 128
#define ABS(x) ((x) >= 0 ? (x) : -(x))

x -> y – 7
ABS(x) -> ((y – 7) >= 0 ? (y – 7) : -(y – 7))

x -> a-- ?
```

Препроцессор и условная компиляция

- Препроцессор позволяет организовать условное включение фрагментов кода в программу

#ifdef name / #endif – проверка определения имени

```
#ifndef _STDIO_H
#define _STDIO_H
<... текст файла ...>
#endif
```

Препроцессор и условная компиляция

- Препроцессор позволяет организовать условное включение фрагментов кода в программу

#if/#if defined/#elif/#else/#endif – общие проверки условий

```
#if HOST_BITS_PER_INT >= 32
typedef unsigned int gfc_char_t;
#elif HOST_BITS_PER_LONG >= 32
typedef unsigned long gfc_char_t;
#elif defined(HAVE_LONG_LONG)
      && (HOST_BITS_PER_LONGLONG >= 32)
typedef unsigned long long gfc_char_t;
#else
#error "Cannot find an integer type with at least 32 bits"
#endif
```

Препроцессор: операции # и

- Операция # позволяет получить строковое представление аргумента

```
#define FAIL(op)                                                \
          do {                                                  \
            fprintf (stderr, "Operation " #op "failed: "        \
                     "at file %s, line %d\n", __FILE__,         \
                      __LINE__);                                \
            abort ();                                           \
          } while (0)

int foo (int x, int y) {
  if (y == 0)
    FAIL (division);
  return x / y;
}

do { fprintf (stderr, "Operation " "division" "failed: " "at file
%s, line %d\n", "fail.c", 13); abort (); } while (0);
```

Препроцессор: операции # и

- Операция ## позволяет объединить фактические аргументы макроса в одну строку

java-opcodes.h:

```
enum java_opcode {
#define JAVAOP(NAME, CODE, KIND, TYPE, VALUE)
       OPCODE_##NAME = CODE,
#include "javaop.def"
#undef JAVAOP
LAST_AND_UNUSED_JAVA_OPCODE
};
```

javaop.def:

```
JAVAOP (nop,              0, STACK,   POP,    0)
JAVAOP (aconst_null,      1, PUSHC,   PTR,    0)
JAVAOP (iconst_m1,        2, PUSHC,   INT,    -1)
<...>
JAVAOP (ret_w,          209, RET,     RETURN, VAR_INDEX_2)
JAVAOP (impdep1,        254, IMPL,    ANY,    1)
JAVAOP (impdep2,        255, IMPL,    ANY,    2)
```

Препроцессор: операции # и

- Операция ## позволяет объединить фактические аргументы макроса в одну строку

```
gcc –E java-opcodes.h:
enum java_opcode {
OPCODE_nop = 0,
OPCODE_aconst_null = 1,
OPCODE_iconst_m1 = 2,
OPCODE_iconst_0 = 3,
<...>
OPCODE_impdep2 = 255,
LAST_AND_UNUSED_JAVA_OPCODE
};
```

Компоновщик

- Организовывает слияние нескольких объектных файлов в одну программу
- Разрешает неизвестные символы (extern переменные и функции)
  - Глобальные переменные с одним именем получают одну область памяти
  - Ошибки, если необходимых имен нет или есть несколько объектов с одним именем
  - Опции для указания места поиска
- Хорошим стилем программирования является экспорт лишь тех объектов, которые используются в других файлах (интерфейс модуля)
  - Используйте квалификатор **`static`**
- Сборка исполняемого файла или библиотеки (*статической* или *динамической*)