Курс «Алгоритмы и алгоритмические языки»
1 семестр 2018/2019

Лекция 13

Приоритеты операций

Операции	Ассоциативность
`( ) [ ] -> .`	Слева направо
`! ~ ++ -- + - sizeof (type) * &`	Справа налево
`* / %`	Слева направо
`+ -`	Слева направо
`<< >>`	Слева направо
`< <= > >=`	Слева направо
`== !=`	Слева направо
`&`	Слева направо
`^`	Слева направо
`|`	Слева направо
`&&`	Слева направо
`||`	Слева направо
`?:`	Справа налево
`= += -= *= /= %= &= ^= |= <<= >>=`	Справа налево
`,`	Слева направо

Объединения

- Объединение – это объект, который может содержать значения различных типов (но не одновременно – только одно в каждый момент)

```
struct constant
{
  int ctype;
  union
  {
    int i;
    float f;
    char *s;
  } u;
} sc;
```

```
switch (sc.ctype)
{
  case CI:
    printf("%d",sc.u.i);
    break;
  case CF:
    printf("%f",sc.u.f);
    break;
  case CS: puts(sc.u.s);
}
```

- Размер объединения достаточно велик, чтобы содержать максимальный по размеру элемент
- Можно выполнять те же операции, что и со структурами

Анонимные объединения и структуры (C11)

- Для вложенных структур и объединений разрешено опускать тег для повышения читаемости

```
struct constant
{
  int ctype;
  union
  {
    int i;
    float f;
    char *s;
  } /* нет имени! */;
} sc;
```

```
switch (sc.ctype)
{
  case CI:
    printf("%d",sc.i);
    break;
  case CF:
    printf("%f",sc.f);
    break;
  case CS: puts(sc.s);
}
```

- Поля анонимной структуры считаются принадлежащими родительской структуре (если родительская также анонимна – то следующей родительской структуре и т.п.)

Битовые поля

- Для экономии памяти можно точно задать размер поля в битах (например, набор флагов)

```
struct tree_base {
  unsigned code : 16;
  unsigned side_effects_flag : 1;
  unsigned constant_flag : 1;
  <...>
  unsigned lang_flag_0 : 1;
  unsigned lang_flag_1 : 1;
  <...>
  unsigned spare : 12;
}
```

  - Адрес битового поля брать запрещено
  - Можно объявить анонимные поля (для выравнивания)
  - Можно объявить битовое поле ширины 0 (для перехода на следующий байт)

Перечисления

- Перечисления – целочисленные типы данных, определяемые программистом.
- Определение перечисления:

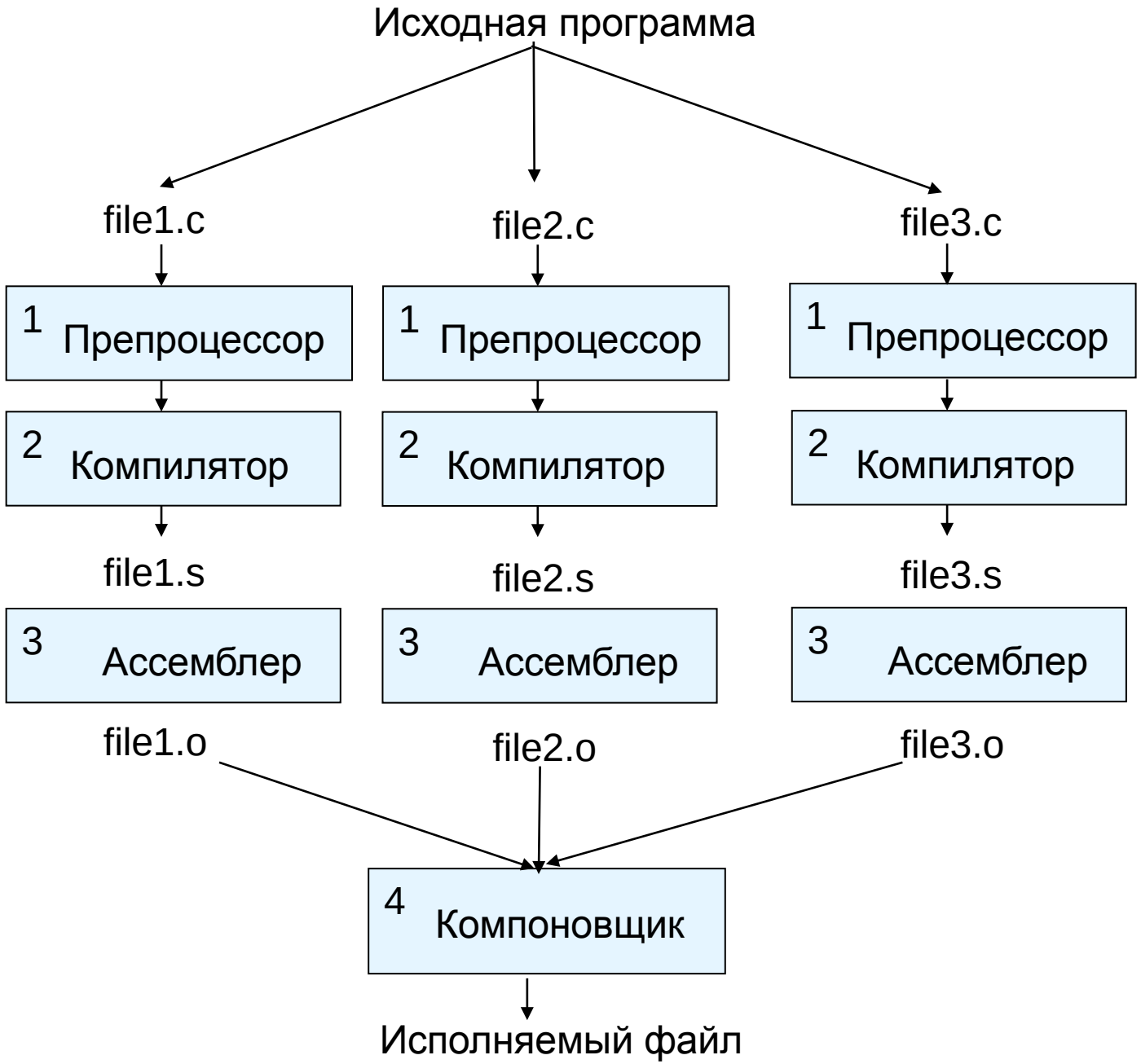
```
enum имя_типа { имена значений };
enum colors {red, orange, yellow, green,
azure, blue, violet};
```

- Значения перечисления нумеруются с 0, но можно присваивать свои значения

```
enum {red, orange = 23, yellow = 23,
green, cyan = 75, blue = 75, violet};
```

- Доступны операции над целочисленными типами и объявление указателей на переменные перечислимых типов
- Проверка корректности присваиваемых значений не производится

Схема раздельной 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
```