

**Курс «Алгоритмы и алгоритмические языки»
1 семестр 2013/2014**

Лекция 12

Динамическое распределение памяти

◆ Функция

```
void *malloc (size_t size)
```

выделяет область памяти размером `size` байтов и возвращает указатель на выделенную область памяти.

Если память не выделена (например, в системе не осталось свободной памяти требуемого размера), возвращаемый указатель имеет значение `NULL`.

◆ Поскольку результат операции `sizeof` имеет тип `size_t` и равен длине операнда в байтах, в качестве `size` можно использовать результат операции `sizeof`.

◆ Тривиальные примеры:

(1) Выделение непрерывного участка памяти объемом 1000 байтов:

```
char *p;  
p = (char *) malloc (1000);
```

(2) Выделение памяти для 50 целых:

```
int *p;  
/* явное приведение типа необязательно */  
p = malloc (50 * sizeof (int));
```

Динамическое распределение памяти

◆ Функция

```
void free (void *p)
```

возвращает системе выделенный ранее участок памяти с указателем `p`.

◆ **Важное замечание:** Аргументом функции `free()` обязательно должен быть указатель `p` на участок памяти, выделенной ранее функцией `malloc()`.

Вызов функции `free()` с неправильным указателем не определен и может привести к разрушению системы распределения памяти.

Вызов функции `free()` с указателем `NULL` не приводит ни к каким действиям (C99).

◆ Функции `malloc()` и `free()` входят в состав библиотеки `stdlib.h`.

Динамическое распределение памяти

◇ *Пример.* Динамическое выделение памяти для строки:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main (void) {
    char *s;
    int t;

    s = (char *) malloc (80 * sizeof (char));
    if (!s) {
        fprintf (stderr, "требуемая память не выделена.\n");
        return 1; /* исключительная ситуация */
    }
    fgets (s, 80, stdin); s[strlen (s) - 1] = '\0';
    /* посимвольный вывод перевернутой строки на экран */
    for (t = strlen(s) - 1; t >= 0; t--)
        putchar (s[t]);
    free (s);
    return 0;
}
```

Динамическое распределение памяти

◆ *Пример.* Динамическое выделение памяти для двумерного целочисленного массива (матрицы):

```
#include <stdio.h>
#include <stdlib.h>
long pwr (int a, int b) {
    long t = 1;
    for (; b; b--) t *= a;
    return t;
}

int main (void) {
    long *p[6];
    int i, j;
    for (i = 0; i < 6; i++)
        if (!(p[i] = malloc (4 * sizeof (long)))) {
            printf ("требуемая память не выделена. \n");
            exit(1);
        }
    for (i = 1; i < 7; i++)
        for (j = 1; j < 5; j++)
            p[i - 1][j - 1] = pwr (i, j);
    for (i = 1; i < 7; i++) {
        for (j = 1; j < 5; j++)
            printf ("%10ld ", p[i - 1][j - 1]);
        printf ("\n");
    }
    /* Цикл с освобождением памяти */
    return 0;
}
```

Динамическое распределение памяти

◆ *Пример.* Динамическое выделение памяти для двумерного целочисленного массива (матрицы):

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

long pwr (int a, int b) {
    long t = 1;
    for (; b; b--)
        t *= a;
    return t;
}

int main (void) {
    long *p[6];
    int i, j;
    for (i = 0; i < 6; i++)
        if (!(p[i] = malloc (4 * sizeof (long)))) {
            printf ("требуемая память не выделена. \n");
            exit(1);
        }
}
```

Динамическое распределение памяти

◇ *Пример.* Динамическое выделение памяти для двумерного целочисленного массива (матрицы):

```
for (i = 1; i < 7; i++)
    for (j = 1; j < 5; j++)
        p[i - 1][j - 1] = pwr (i, j);
for (i = 1; i < 7; i++) {
    for (j = 1; j < 5; j++)
        printf ("%10ld ", p[i - 1][j - 1]);
    printf ("\n");
}
for (i = 0; i < 6; i++)
    free (p[i]);
return 0;
}
```

VLA-массивы

- ◇ В Си-89 размер массива обязан являться константой. Это неудобно при передаче массивов (многомерных) в функции:

```
/* можно передать int a[5]; int a[42]; ... */
```

```
int asum1d (int a[], int n) {
```

```
    int s = 0;
```

```
    for (int i = 0; i < n; i++)
```

```
        s += a[i];
```

```
    return s;
```

```
}
```

```
/* можно передать только int a[???][5] */
```

```
int asum2d (int a[][5], int n) {
```

```
    int s = 0;
```

```
    for (int i = 0; i < n; i++)
```

```
        for (int j = 0; j < 5; j++)
```

```
            s += a[i][j];
```

```
    return s;
```

```
}
```


VLA-массивы

- ◇ В Си-99 размер массива автоматического класса памяти может задаваться во время выполнения программы:

```
int foo (int n) {  
    int a[n];  
    <... Можно обрабатывать a[i]...>  
}  
/* можно передать int a[???][???] */  
int asum2d (int m, int n, int a[m][n]) {  
    int s = 0;  
    for (int i = 0; i < m; i++)  
        for (int j = 0; j < n; j++)  
            s += a[i][j];  
    return s;  
}
```

- ◇ Объявление функции asum2d:

```
int asum2d (int m, int n, int a[m][n]);  
int asum2d (int, int, int [*][*]);
```

VLA-массивы и динамическое выделение памяти

- ◆ Функция `asum2d` может использоваться с VLA-массивами, но они всегда выделяются в автоматической памяти:

```
int foo (int m, int n) {  
    int a[m][n]; int s;  
    <... Считаем a[i][j]...>  
    s = asum2d (m, n, a);  
}
```

- ◆ Можно выделить VLA-массив в динамической памяти:

```
int main (void) {  
    int m, n;  
    scanf ("%d%d", &m, &n);  
    int (*pa)[n];  
    pa = (int (*)[n]) malloc (m * n * sizeof (int));  
    <... Считаем pa[i][j]...>  
    s = asum2d (m, n, pa);  
    free (pa);  
}
```

Динамическое распределение памяти

- ◇ Состав функций динамического распределения памяти библиотеки `stdlib` (заголовочный файл `<stdlib.h>`)

```
void *malloc (size_t size);  
void free (void *p);  
void *realloc (void *p, size_t size);  
void *calloc(size_t num, size_t size);
```

- ◇ Функция `void *realloc (void *p, size_t size)` согласно стандарту Си99 сначала выполняет `free (p)`, а потом `p = malloc (size)`, возвращая новое значение указателя `p`. При этом значения первых `size` байтов новой и старой областей совпадают.

- ◇ Функция `void *calloc (size_t num, size_t size)` работает аналогично функции `malloc (size1)`, где `size1 = num * size` (т.е. выделяет память для размещения массива из `num` объектов размера `size`).

Выделенная память инициализируется нулевыми значениями

Массив переменного размера в структуре (C99)

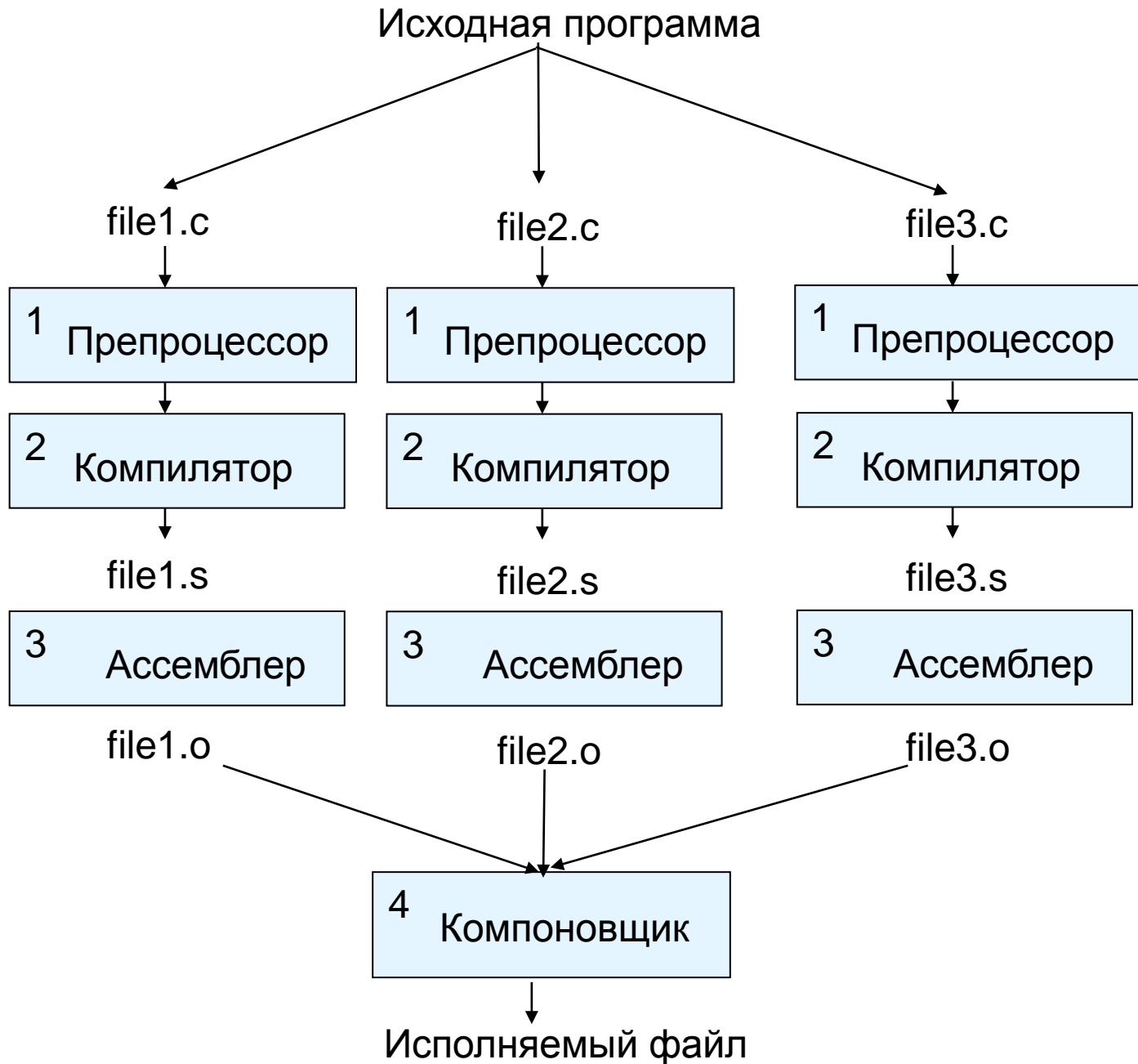
- ◆ Flexible array member – последнее поле структуры:

```
struct polygon {  
    int np; /* число вершин */  
    struct point points[];  
}
```

- ◆ Варьирование размера переменного массива:

```
int np; struct polygon *pp;  
scanf ("%d", &np);  
pp = malloc (sizeof (struct polygon)  
             + np * sizeof (struct point));  
pp->np = np;  
for (int i = 0; i < np; i++)  
    scanf ("%d%d", &pp->points[i].x,  
          &pp->points[i].y);
```

Схема компиляции



Преппроцессор

- ◇ 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);
```