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

Лекция 14

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
};
```

Компоновка и классы памяти

Класс памяти	Время жизни	Видимость	Компоновка	Определена
автоматический	автоматическое	блок	нет	В блоке
регистровый	автоматическое	блок	нет	В блоке как **`register`**
статический	статическое	файл	внешняя	Вне функций
статический	статическое	файл	внутренняя	Вне функций как **`static`**
статический	статическое	блок	нет	В блоке как **`static`**

- Квалификатор **`extern`**: переменная определена и память под нее выделена в другом файле
- Классы памяти функций:
  - статическая (объявлена с квалифатором **`static`**)
  - внешняя (**`extern`**), по умолчанию
  - встраиваемая (**`inline`**, C99)
- Объявление внешних функций в заголовочных файлах:

```
extern void *realloc (void *ptr, size_t size);
```

Компоновщик

- Организовывает слияние нескольких объектных файлов в одну программу
- Разрешает неизвестные символы (внешние 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*)

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 * sizeof (char));
```

  (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).

  Обращение к освобожденному указателю не определено.

- 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 двухмерного 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;
}
```

Динамическое распределение памяти

- ***Пример.*** Динамическое выделение памяти для двухмерного целочисленного массива (матрицы):

```
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");
  }
  for (i = 0; i < 6; i++)
    free (p[i]);
  return 0;
}
```