Основы алгоритмизации и программирования. Обработка ошибок в Си

Лекция 12

Привезенцев Д.Г.

Муромский институт Владимирского государственного университета
Очная форма обучения

2 декабря 2021 г.

Возврат специальных значений

В языке C при ошибке функции возвращают специальные значения или устанавливают флаги ошибки.

О том,что делает функция в такой ситуации и как сообщает об этом, указывают в документации.

Подробная информация о каждой функции и библиотеке можно найти в
C standard library reference
`https://www.tutorialspoint.com/c_standard_library/index.htm`

Пример возвращаемого заначения scanf

Листинг 1: Контроль введенных значений функцией scanf

```
#include <stdio.h>
#include <errno.h>
int main()
{
  int a, b;
  int items = scanf("%d%d", &a, &b);
  if(items == 2)
  {
    printf("Success! a=%d, b=%d", a, b);
    // тут можно использовать a и b
  }
  else if(items == 1)
  {
    printf("Error! a=%d, b is undefined", a);
    // тут можно использовать только a
  }
  else
  {
    printf("Error! a and b are undefined");
  }
  getchar();
  getchar();
  return 0;
}
```

Неопределённое поведение

Игнорирование ошибок в некоторых случаях можетпривести к совершению программой некорректных действий.
По стандарту в C некоторые операции (например, выход за границу массива, разыменование нулевого указателя, переполнение знаковых целых) ведут к неопределённому поведению.

Неопределённое поведение

Игнорирование ошибок в некоторых случаях можетпривести к совершению программой некорректных действий.
По стандарту в C некоторые операции (например, выход за границу массива, разыменование нулевого указателя, переполнение знаковых целых) ведут к неопределённому поведению.

Например, при выходе за границы массива можно попытаться перезаписать либо «чужую» память, либо «свои» переменные, либо пустую область в «своей» памяти.
В первом случае программа сразу завершится с ошибкой,
во втором – возникнут ошибки в случайных местах программы,
в третьем – ошибка может не проявляться в течение длительного времени.

В случае деления на ноль (или малые числа) чисел с плавающей точкой обычно возникают значения «не число» (**NaN**, **not a number**) или «бесконечность», пригодные для дальнейших вычислений; в случае целых чисел деление на ноль создаёт исключительную ситуацию

Листинг 2: Деление на ноль

```
#include <stdio.h>
int main()
{
  double a = 4, b = 0;
  printf("%lf\n", a/b);

  int c = 4, d = 0;
  printf("%d\n", c/d);

  getchar();
  return 0;
}
```

Глобальное значение **errno**

В заголовочном файле <errno.h> объявляется целое значение errno, имеющее смысл индикатора и кода ошибки. При запуске про-граммы оно обнуляется, а функции из стандартной библиотеки могут его изменять, устанавливая ненулевые значения

Листинг 3: Ошибка при отрытии файла

```
#include <stdio.h>
#include <errno.h>
int main()
{
  errno = 0;
  FILE *f = fopen("file.txt", "r");
  if(f == NULL)
  {
    int errno_value = errno;
    printf("%d\n", errno_value);
    //do_something(); // действие для всех ошибок
    switch(errno_value)
    {
      // действия для конкретного типа ошибки
    }
  }
}
```

Стандартный поток ошибок

В программе существует три стандартных потока. Двумя из них – стандартным потоком ввода (**stdin**) и стандартным потоком вывода (**stdout**) – скорее всего уже пользовался каждый, работая с scanf и printf. Третий – стандартный поток ошибок (**stderr**) – предназначен для вывода различных диагностических сообщений и сообщений об ошибках.

Листинг 4: Вывод ошибок в спец. поток

```
#include <stdio.h>
#include <errno.h>
int main()
{
  errno = 0;
  FILE *f = fopen("file.txt", "r");
  if(f == NULL)
  {
    int errno_value = errno;
    fprintf(stderr, "Ошибка с кодом %d.\n", errno_value);
    //do_something(); // действие для всех ошибок
    switch(errno_value)
    {
      // действия для конкретного типа ошибки
    }
  }
}
```

Возврат значения по умолчанию (например, функция преобразования строки в число **strtol** возвращает ноль, если не может осуществить преобразование).

```
int fac(int x)
{
    if(x < 0)
        return 1;
    ...
}
```

Возврат специального значения (например, функция **getc** читает символ, но возвращает значение более общего типа int, вмещающего также **EOF**, которое возвращается в случае ошибки)

```
float* get_vector(int dim)
{
    float* v = (float*) malloc(dim * sizeof(float));
    if(v == NULL)
        return NULL;
    ...
}
```

Установка **errno** или флага ошибки из другой глобальной переменной (например, упомянутая **strtol** изменяет **errno**, если не может осуществить преобразование).

Установка локального флага ошибки, переданного по указателю.

В отличие от установки глобальных флагов, можно исключить перезапись флага другими функциями; также проясняется обработка ошибок для рекурсивных функций. В отличие от возврата специального значения, ошибка не разделяет область значений функции с её результатом.

В этом случае программисту приходится явно передавать указатель на флаг, а значит есть вероятность, что он не забудет обработать ошибку.

```
int fac(int x, int * error)
{
    if(x < 0)
        *error = 1;
    ...
}
```

Вывод сообщения об ошибке в **stderr**, файл или базу данных. Не является необходимой и достаточной мерой при обработке ошибок, но помогает человеку впоследствии проанализировать поведение программы. Стоит помнить, что сообщение никак не поможет узнать о наличии ошибки коду, который вызвал функцию.

Выход из программы. Самый простой сценарий, исключающий дальнейшую обработку ошибок

```
void* malloc_strict(size_t size)
{
    void* p = malloc(size);
    if(p == NULL)
    {
        fprintf(stderr, "Cannot allocate memory");
        exit(EXIT_FAILURE);
    }
    ...
}
```