1 Занятие №4

1.1 Идентификаторы в Си

Идентификаторы начинаются с латинской буквы (большой или маленькой) или знака подчёркивания, далее идут латинские буквы, цифры или знаки подчёркивания. Длина идентификатора неограничена, отдельные компиляторы могут накладывать ограничения на максимальное число значащих символов. На внешние имена тоже могут накладываться ограничения. *Регистр букв является значимым*, то есть `a` и `A` — это два разных идентификатора. Некоторые идентификаторы зарезервированы для использования в качестве ключевых слов, например **`int`**, **`do`**. В программах не рекомендуется определять и использовать идентификаторы, начинающиеся со знака подчёркивания. Такие идентификаторы зарезервированы для внутреннего использования стандартными библиотеками.

1.2 Литеральные значения

1.2.1 Целые

Целые значения могут записываться в программе в десятичной, восьмеричной и шестнадцатеричной системе счисления. *Восьмеричная константа* начинается с символа `0` («ноль»), за которым идут цифры `0`—`7`, например `0377`. *Шестнадцатеричная константа* начинается с символов `0x`, затем идут цифры `0`—`7`, `a`—`f`, `A`—`F`. Пример `0xFF`. *Десятичная константа* начинается с цифр `1`—`9`, далее идут цифры `0`—`9`.

Тип константы — это минимальный тип, который может содержать данное значение. Для констант, заданных в десятичной форме, последовательно выбираются **`int`**, **`long`**, **`unsigned long`**, **`long long`**, **`unsigned long long`** (для C99). Для констант, заданных в восьмеричной или шестнадцатеричной форме, последовательно выбираются **`unsigned int`**, **`unsigned long`**, **`unsigned long long`** (для C99).

Тип константы можно задать явно указанием суффикса `u` или `l`. Эти суффиксы могут употребляться совместно. Например, `10u` — константа 10 типа **`unsigned int`**, `6uL` — константа 6 типа **`unsigned long`**. `7ll` — константа 7 типа **`long long`**.

1.2.2 Вещественные

Вещественная константа может содержать целую и дробную часть мантиссы и порядок. Вещественная константа записывается в десятичной системе счисления. По умолчанию вещественные константы имеют тип **`double`**. Для явного задания типа можно использовать суффикс `l` или `L` для указания типа **`long double`**. Суффикс `f` или `F` для указания типа **`float`**.

1.2.3 Литерные

Литерные константы имеют тип **`int`**. В апострофах может быть записано несколько символов, в этом случае соответствующее целое значение зависит от компилятора.

1.2.4 Строковые

Строковые литералы имеют тип **`char const *`** (константный указатель на тип **`char`**). Все строковые константы имеют в конце строки неявный символ с кодом `'\0'` — термина-

тор строки. Строка не имеет явного поля длины, в отличие от языка Turbo Pascal.

1.3 Простейшая работа со строками

Строки — это последовательность символов. Строки могут храниться в массивах типа **char**, **signed char** или **unsigned char**. Переменная для хранения строки определяется как массив

```
char str[N];
```

где N задаёт объем памяти, отводимый для строки. Поскольку строка всегда хранит символ-терминатор '\0', максимальное число значащих символов в такой строке на единицу меньше (N-1 символ).

Строки можно инициализировать следующим образом:

```
char str[20] = "a string";
```

В этом примере будет задано значение 9 байт (8 значащих символов и один символ-терминатор), остальные 11 байт будут либо обнулены, либо содержать произвольное значение.

Символьные массивы можно определять без указания размера, если присутствует инициализация. Например,

```
char x[] = "xxx";
```

В этом случае под массив x будет выделено 4 байта.

Для манипуляций со строками в языке Си используются стандартные библиотечные функции. Чтобы их можно было использовать, в начале программы нужно подключить заголовочный файл string.h.

```
#include <string.h>
```

Некоторые из этих функций мы рассмотрим.

Функция strcmp, прототип которой упрощённо записывается

```
int strcmp(char s1[], char s2[]);
```

сравнивает две строки s1 и s2. Если строка s1 лексикографически меньше строки s2, функция возвращает отрицательное значение. Если строка s1 лексикографически больше строки s2, функция strcmp возвращает положительное значение. Если две строки равны, функция возвращает ноль. Проверка на равенство двух строк записывается несколько неожиданным способом

```
if (!strcmp(s1, s2)) { /* ... */ }
```

Обратите внимание, что *строки нельзя сравнивать обычными операциями сравнения* !=, == и т. д. Дело в том, что в этом случае будут сравниваться не значения строк, а адреса этих строк. Компилятор в этом случае не даст никаких предупреждений.

Функция strlen с прототипом

```
int strlen(char s[]);
```

вычисляет количество значащих символов в строке, то есть число символов до символа-терминатора. Например strlen("xx") даёт результат 2.

Функция strcpy с прототипом

```
char *strcpy(char dst[], char src[]);
```

копирует строку str в строку dst, включая символ-терминатор. Как и везде в языке Си, контроль переполнения массива полностью лежит на программисте. Существует вариант этой функции (strncmp), который позволяет ограничить максимальное количество копируемых символов.

Функция strcat с прототипом

```
char *strcat(char dst[], char src[]);
```

добавляет строку str в конец строки dst. Существует вариант этой функции (strncat), который позволяет ограничить число добавляемых символов.

Строку очистить можно с помощью функции strcpy

```
strcpy(s, "");
```

а можно проще:

```
s[0] = 0;
```

1.4 Ввод/вывод строк

Для вывода строк предусмотрен специальный спецификатор формата %s который можно использовать в функциях семейства printf. Например,

```
printf("His␣name␣is␣%s\n", name);
```

Символ-терминатор на печать выводиться не будет.

Считывать строки можно аналогичным спецификатором формата %s для функций семейства scanf. Например,

```
char buf[20];
scanf("%s", buf);
```

В этом случае сначала будут пропущены все пробельные символы, затем все символы до первого пробельного символа будут занесены в buf. После этого в buf будет добавлен символ-терминатор '\0'. Обратите внимание на отсутствие знака '&' перед buf в аргументах вызова scanf. Буфер buf должен быть достаточного размера, чтобы вместить все символы вводимой строки. Поскольку вводимые данные чаще всего не находятся под контролем программиста, то есть пользователь, возможно злонамеренно, может вводить строки произвольной длины, *использование такого неконтролируемого чтения в реальных программах очень опасно*.

Другой полезный спецификатор для scanf — %n. Этому спецификатору должна соответствовать переменная целого типа со знаком & (адрес). В эту переменную заносится число символов, прочитанное к моменту, когда был встречен этот спецификатор. Спецификатор %n не учитывается в числе успешно считанных спецификаторов в возвращаемом значении функции scanf.

Очень полезной является функция sprintf с прототипом

```
int sprintf(char s[], char format, ...);
```

Которая работает точно также, как printf, поддерживает все форматы вывода, но вместо печати на стандартный поток вывода, заполняет строку s. В конец строки s добавляется символ-терминатор. Результатом работы функции является количество выведенных символов (не считая символ-терминатор). Например,

```
sprintf(s, "%d", n);
```

позволяет получить в строке s символьное представление числа n.

«Обратная» функция sscanf с прототипом

```
int sscanf(char s[], char format, ...);
```

считывает из форматные данные из строки вместо стандартного потока ввода. Функция возвращает число успешно считанных полей (как и scanf). Например,

```
sscanf(s, "%d", &n);
```

считывает в переменную n число из строки s.

Слово «строка» в русском языке может обозначать два совершенно разных понятия. Во-первых, это просто цепочка символов (string); во вторых, это последовательность символов, занимающая один ряд на экране или в напечатанном тексте (line). В первом случае иногда говорят «цепочка».

Очень часто бывает так, что стандартный поток ввода ассоциирован либо с терминалом, либо с файлом, который имеет текстовую структуру, то есть разбит на строки (line). В языке Си входной поток является последовательностью символов. Строки текстового файла во входном потоке разделяются символом '\n', не зависимо от того, какой разделитель строк текста в действительности используется в операционной системе (например, в MS-DOS используются два символа '\r', '\n'). Входной поток не имеет специального символа-признака конца входного потока. Таким образом, входной файл вида

```
a
b b
cc
```

будет представлен как поток символов

```
'a', '\n', 'b', ' ', 'b', '\n', 'c', 'c', '\n', '\n'
```

Функция gets с прототипом

```
char * gets(char s[]);
```

считывает одну строку входного текстового файла, то есть последовательность символов до символа '\n', включая его. Считанные символы помещаются в строку s, причём символ '\n' заменяется на символ-терминатор '\0'. В случае, если достигнут конец входного потока, функция возвращает специальную константу NULL. Обратите внимание, что когда входной поток ассоциирован с терминалом, для того, чтобы был отмечен конец потока, нужно нажать специальную комбинацию клавиш. Буфер s должен быть достаточного размера, чтобы вместить всю считываемую последовательность символов. *Использование этой функции опасно.*

Функция puts с прототипом

```
char *puts(char s[]);
```

добавляет в выходной поток символы из строки s. Символ-терминатор заменяется на символ завершения строки '\n'.

1.5 Побочный эффект в операциях инкремента и декремента

Основной эффект операции — это выработка некоторого значения, которое может быть далее использовано в выражении. Побочный эффект — это другое воздействие на среду выполнения (например, изменение значения переменной).

Нами уже были рассмотрены операции инкремента (увеличения на 1) переменной `++`, и декремента (уменьшения на 1) переменной `--`. В языке Си они существуют в двух формах: префиксной и постфиксной, которые отличаются основным эффектом операции. Преинкремент `++a` — значение переменной вначале увеличивается на 1, и это уже увеличенное значение является значением операции, и далее в выражении может быть использовано. Постинкремент `a++` — значением этой операции является значение переменной до увеличения на 1. Например,

```
int a = 5, b, c;
b = 5 + ++a;
c = 5 + a++;
```

значением переменной `b` будет 11, а переменной `c` — 10.

1.6 Операторы передачи управления

1.6.1 Оператор `break`

Оператор **`break;`** передаёт управление за пределы самого вложенного оператора **`do`**, **`while`**, **`for`** или **`switch`**. Оператор **`break;`** не может использоваться вне этих операторов. Например,

```
switch (x) {
  case a:
    while (1) {
      if (y) break;
    }
    break;
  default:
    ;
}
```

1.6.2 Оператор `continue`

Оператор **`continue;`** передаёт управление за оператор, составляющий тело цикла **`do`**, **`while`** или **`for`**, что означает немедленное начало следующей итерации цикла. Для оператора **`do`** произойдёт переход на вычисление управляющего выражения **`while`** в конце цикла, для оператора **`while`** произойдёт переход на вычисление управляющего выражения в начале цикла. Для оператора **`for`** произойдёт переход к вычислению третьего выражения заголовка цикла **`for`**.

1.7 Упражнения

1. Написать функцию, которая реверсирует строку, переданную в качестве параметра.

2. Написать функцию, которая в массиве из целых чисел все числа, меньшие или равные заданному переставляет в начало массива, а все числа, большие или равные заданному — в конец массива.

3. Проверить на равенство строки (функция strcmp).

4. Скопировать из входного потока в выходной все строки, длина которых больше 20.

5. Скопировать из входного потока в выходной все строки, которые содержат целое число, большее 666.