Курс лекций "Программирование"
Основы программирования на языках C и C++

Лекция 6. Некоторые функции библиотеки C

Глухих Михаил Игоревич, к.т.н., доц.

mailto: glukhikh@mail.ru

Заголовочные файлы библиотеки C

- <stdio.h> -- стандартный ввод-вывод: printf, scanf, fopen, fprintf, fread, fwrite, …
- <math.h> -- математика: sqrt, sin, cos, …
- <string.h> -- основные строковые функции: strlen, strcpy, …
- <stdlib.h> -- динамическая память, генератор случайных чисел, некоторые строковые функции: malloc, free, rand, atoi, …
- <time.h> -- время: time, …

Форматированный ввод-вывод

- Подключаемый файл <stdio.h>
  - printf – вывод на консоль
  - scanf – ввод с консоли
  - fprintf – вывод в файл
  - fscanf – ввод из файла
  - sprintf - создание строки
  - sscanf – анализ строки
  - gets, fgets – ввод строк
  - getc, fgetc, getchar – ввод символов

Форматированный вывод на консоль

```
printf(строка_формата, аргумент1,
    аргумент2, ..., аргументN)
// Простой случай (без аргументов)
printf("Hello, world!\n");
// Сложный случай (с аргументами)
int a=2, b=3;
int sum=a+b;
printf("%d=%d+%d", sum, a, b);
```

Обработка строки формата

- В обычном случае строка формата просто выводится на экран
- Модификаторы формата, начинающиеся с символа %, заменяются очередным аргументом
- В модификаторах формата задается предполагаемый тип аргумента (фактически он **не проверяется**) и способ его вывода

Простые модификаторы формата

- %c – подстановка **char**
- %d или %i – подстановка **int**
  - %o – то же в восьмеричной системе
  - %x – то же в шестнадцатеричной системе
- %u – подстановка **unsigned**
- %f, %lf – подстановка **float, double** в обычной форме
  - аналог cout<<fixed
- %e, %le – то же в экспоненциальной форме
  - аналог cout<<scientific
- %g, %lg – то же в форме, занимающей меньше места
- %s – подстановка строки (**char***)
- %% – вывод символа % (не является модификатором формата, так как не приводит к чтению аргумента)

Сложные модификаторы формата

- Все сложные модификаторы формата влияют только на вывод одной переменной (в отличие от манипуляторов для cout<<)
- %5d – вывод целого числа в 5 позициях
  - аналог cout<<setw(5)
- %05d – вывод целого числа в 5 позициях с нулями впереди
  - аналог cout<<setfill('0')<<setw(5)
- %9.3lf – вывод вещественного числа (**double**) в 9 позициях с 3 знаками после десятичной точки
  - аналог cout<<setw(9)<<setprecision(3)
- %+5d или %+9.3lf – вывод числа со знаком (+ или -) впереди
  - аналог cout<<showpos
- %-5d – вывод числа с выравниванием влево (число слева, пробелы справа)
  - аналог cout<<left
- по умолчанию все выравнивается вправо

Сложные модификаторы формата (строки)

- %20s – вывод строки в 20 позициях (по умолчанию строка выравнивается вправо, слева вставляются пробелы)
- %-20s – вывод строки в 20 позициях с выравниванием влево
- %.5s – вывод первых 5 символов строки
- %20.5s – вывод первых 5 символов строки в 20 позициях

Пример

```
// Переменная currTime содержит время
// в минутах, необходимо вывести hh:mm
// С помощью cout
cout<<setfill('0');
cout<<setw(2)<<currTime/minInHour<<":";
cout<<setw(2)<<currTime%minInHour;
// С помощью printf
printf("%02d:%02d", currTime/minInHour,
    currTime%minInHour);
```

Форматированный ввод с клавиатуры

```
scanf(строка_формата, аргумент1, аргумент2,
    ..., аргументN)
```

- Результат функции – число успешно прочитанных аргументов
- Для целых, вещественных, символьных аргументов используются **адреса** (&a, &f, &c)

```
// Например
int a, b;
scanf("%d%d", &a, &b);
```

Форматированный ввод с клавиатуры

- Для строк используется адрес первого элемента (или имя массива)

```
// Например
char str[20];
// Если пользователь введет 20 и более
// символов, то произойдет ошибка
scanf("%s", str);
// или (запись в массив
// начиная с 5-го элемента)
scanf("%s", &(str[5]);
```

Анализ строки формата

- Символы – не модификаторы формата – ожидаются во вводимой строке
- Простые модификаторы формата те же, что для printf (%c, %d, %i, %o, %x, %f, %lf, %e, %le, %g, %lg, %s)
- %20s – ввод не более 20 символов строки (безопасно, в отличие от %s)
- %3d – ввод целого числа не более 3 цифр
- %*lf – прочитать вещественное число, но не записывать в очередной аргумент

Достоинства <stdio.h>

- printf и scanf присутствуют и в реализациях чистого C, и в реализациях C++; cout<< и cin>> присутствуют только в реализациях C++
- В случаях, когда необходим сложный форматированный ввод/вывод, с помощью scanf/printf запись получается более компактной
- В целом, число возможностей scanf/printf несколько больше (см. также MSDN)
- Строки формата встречаются и в других языках – например, Java, Python

Недостатки <stdio.h>

- printf и scanf **не контролируют** типы аргументов; если тип аргумента не соответствует ожидаемому, в лучшем случае произойдет ошибка, а в худшем произойдет ввод или вывод не той информации, которую ожидает программист
- printf и scanf не могут выводить переменные пользовательских типов; с помощью cout<< и cin>> это возможно
- На мой взгляд, ввод-вывод с помощью cin>> и cout<< нагляднее и проще в освоении
- Но ввод-вывод с помощью printf/scanf часто компактнее

Простой ввод строк / символов

```
#include <stdio.h>
// Один символ
int fgetc(FILE *stream);
int getc(FILE *stream);
int getchar(void);
// Строка (безопасно)
char* fgets(char* s, int size,
            FILE* stream);
// Строка (опасно, не использовать)
char* gets(char* s);
```

Форматированный вывод в файл

```
fprintf(дескриптор_файла, строка_формата,
    аргумент1, аргумент2, …, аргументN);
// Для получения дескриптора файла
// файл нужно открыть
FILE* fout=fopen("out.txt", "w");
if (fout==0) { … } // ошибка открытия
fprintf(fout, "%d=%d+%d",a+b,a,b);
// По окончании работы с дескриптором
// файл нужно закрыть
// (для ofstream это не требуется)
fclose(fout);
```

Концепция ресурсов и владения ресурсами

- Динамическая память, дескриптор файла, ... – это всё примеры ресурсов
- Характерный признак – необходимость явного создания и явного разрушения ресурса
- Владелец ресурса – по умолчанию тот, кто его создал (функция)
- Владелец ресурса должен сам его освободить (стандартный вариант)
- Либо передать владение кому-то ещё (пример: функция чтения матрицы)

Неформатированный вывод в файл

```
int arr[20];
// Заполнение array
// …
FILE* fout=fopen("out.txt", "wb");
if (fout==0) { … }
// Вывод массива в двоичной форме
fwrite(arr, sizeof(int), 20, fout);
// Закрытие файла
fclose(fout);
```

То же с помощью ofstream

```
int arr[20];
// Заполнение array
// …
ofstream out("out.txt", ios::binary);
if (!out.is_open()) { … }
// Вывод массива в двоичной форме
out.write((char*)arr, sizeof(arr));
```

Форматированный ввод из файла

```
fscanf(дескриптор_файла, строка_формата,
    аргумент1, аргумент2, …, аргументN)
// Для получения дескриптора файла
// файл нужно открыть
FILE* fin=fopen("out.txt", "r");
if (fin==0) { … } // ошибка открытия
fscanf(fin, "%d %d", &a, &b);
// По окончании работы с дескриптором
// файл нужно закрыть
// (для ifstream это не требуется)
fclose(fin);
```

Неформатированный ввод из файла

```
int arr[20];
FILE* fin=fopen("in.txt", "rb");
if (fin==0) { … }
// Ввод массива в двоичной форме
fread(arr, sizeof(int), 20, fin);
fclose(fin);
// Или - то же самое через ifstream
ifstream in("out.txt", ios::binary);
if (!in.is_open()) { … }
// Ввод массива в двоичной форме
in.read((char*)arr, sizeof(arr));
```

Создание строки

```
sprintf(строка, строка_формата, аргумент1,
    аргумент2, …, аргументN);
```

- Работает так же, как и printf, но результат записывается не на экран, а в строку (символьный массив). В массиве должно быть достаточно места.

```
char str[50];
int a=2, b=3;
sprintf(str,"%d=%d+%d",a+b,a,b);
// Результат: 5=2+3 в str
```

Анализ (разбор) строки

```
sscanf(строка, строка_формата,
    аргумент1, аргумент2, …, аргументN);
```

- Работает аналогично fscanf, но чтение информации происходит из строки, а не из файла

```
char str[]="5=2+3";
int a,b,s;
sscanf(str,"%d=%d+%d",&s,&a,&b);
// Будет выведено s=5 a=2 b=3
printf("s=%d a=%d b=%d\n",s,a,b);
```

Deprecated функции

- В MVS 2005/2008 многие функции из <stdio.h> объявлены **deprecated**. Буквально это означает – устаревающие, а фактически – небезопасные. Их использование приводит к предупреждениям (warning) при компиляции
- Существуют более безопасные версии функций с постфиксом _s, например, printf_s, scanf_s и т.д., можно использовать их (данные функции при работе делают большее количество проверок). Функции подробно описаны в MSDN.
- Либо можно указать в начале программы
- #**define** _CRT_SECURE_NO_DEPRECATE
- И предупреждений не будет
- NB: так лучше, так как эти функции с постфиксом _s нестандартные!

Функции выделения и освобождения памяти

```
#include <stdlib.h>

// Выделение size байт
// size_t специальный тип для размеров
void* malloc(size_t size);
// Освобождение памяти
void free(void* ptr);
// Перевыделение памяти другого размера
// с переносом содержимого
void* realloc(void* ptr, size_t size);
```

Что такое void*

- Так называемый нетипизированный указатель или указатель на регион
- Особенность – нельзя разадресовывать
- В языке C void* можно преобразовать к любому другому указателю и обратно (что обычно и делается после вызова malloc)
- В языке C++ void* используется значительно реже, есть ограничения по его преобразованию

Генератор случайных чисел и время

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

// Инициализация ГСЧ, seed -- зерно
void srand(unsigned seed);
// Генерация случайного числа
int rand(void);
// Получение времени в секундах
// от начала компьютерной эпохи
time_t time(time_t* ptr);
```

Пример использования

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

int main(void) {
    srand((unsigned)time(0));
    // …
    int dice = rand() % 6; // 0..5
}
```

Преобразования строки в число и обратно

- Упрощенные версии разбора sscanf / вывода в строку sprintf

```
#include <stdlib.h>

// Строка в число
// Более продвинутая версия -- strtol
int atoi(const char* ptr);
// Строка в число (нестандартная)
char* itoa(int val, char* ptr, int base);
```

Итоги

- Рассмотрены функции C
  - Форматированный ввод-вывод на консоль/в файл/в строку
  - Модификаторы формата
  - Неформатированный ввод-вывод
  - Выделение / освобождение памяти
  - ГСЧ / время
  - Преобразование число ←→ строка
- Далее
  - Функциональная декомпозиция в C и C++