Язык С (часть 2)

Многофайловые программы

Запуск **gcc** позволяет обработать файл с исходным кодом препроцессором и далее скомпилировать его.

Однако при этом сам инструмент **gcc** не компилирует файл исходного кода в конечный исполняемый файл. Он компилирует его в объектный файл, после чего вызывает так называемый ***линковщик***, или компоновщик.

Для программ, состоящих из одного файла, такой необходимости нет. Хотя при желании здесь также можно отказаться от компоновки, если выполнить команду gcc с ключом -c:

gcc -c hello.c

В результате получится файл с расширением *.o. Чтобы получить из объектного файла исполняемый, надо использовать ключ -o:

gcc -o hello hello.o

Для программ, состоящих из нескольких файлов исходного кода, получение объектных файлов является необходимым.

Именно из них потом компонуется единственный исполняемый файл.

Компиляция программы, состоящей из нескольких файлов исходного кода

Рассмотрим пример. Пусть в одном файле определена пара функций, а в другом, содержащем функцию main(), осуществляется их вызов.

Файл **superprint.c:**

```
#include <stdio.h>
void  l2r (int *c, int n) {int i;
for(i=0; i<n; i++)printf ("%d\t",c[i]);
printf("\n");}
void  r2l (int *c, int n) {int j;

for (j=0; j<n; j++)
printf ("%d\n", c[j]);
}
```

Файл **main.c**:

```
#include <stdio.h>
#define N 5
main () {
int A[N];
int i;
for(i=0; i<N; i++)
scanf("%d", &A[i]);
printf("Massiv v stroku\n");
l2r(A, N);
printf("Massiv v stolbec\n");
r2l(p, N);
}
```

Чтобы получить исполняемый файл этой программы, надо сначала получить объектные
файлы из исходных:

gcc -c superprint.c

gcc -c main.c

Тоже самое можно сделать за один вызов gcc:

gcc -c superprint.c main.c

Или даже вот так, если в каталоге находятся только файлы текущего проекта:

gcc -c *.c

В любом случае в каталоге появятся два объектных файла: superprint.o и main.o.

Далее их можно скомпилировать в один исполняемый файл так:

gcc -o myprint main.o superprint.o

или так:

gcc -o myprint *.o

Обратите внимание на то, что в данном случае обязательно требуется указывать имя исполняемого файла.

Такой вариант недопустим:

gcc -o main.o superprint.o

Задумаемся, каким образом в представленной выше программе функция main() "узнает" о существовании функций l2r() и r2l().

Если попытаться получить из main.c отдельный исполняемый файл, т.е. скомпилировать программу без superprint.c:

gcc main.c , то ничего не получиться.

Получить из файла superprint.c отдельный исполняемый файл вообще невозможно, т.к. там **отсутствует** функция **main().**

А вот получить из этих файлов отдельные объектные файлы можно.

Связывание происходит при компиляции исполняемого файла из объектных.

Создание заголовочных файлов

Что будет, если в функции main() осуществить неправильный вызов функций l2r() и r2l()? Например, указать неверное количество параметров.

Создание объектных файлов пройдет без ошибок. Скорее всего удастся получить исполняемый файл; но вот работать программа будет неправильно.

Такое возможно потому, что ничего не контролирует соответствие вызовов прототипам (объявлениям) функций.

Куда правильней сообщать о неверном вызове функций уже при получении объектного файла. Поэтому хотя можно обойтись и без этого, но очень желательно сообщать функции main() прототипы функций, которые из нее вызываются. Это можно сделать, прописав объявления функций в файле **main.c:**

```
void l2r (int *c, int n);
void r2l (int *c, int n);
main () {
…
```

Теперь, если мы передадим неправильные параметры, ошибка возникнет уже на этапе получения объектных файлов.

Все прототипы (объявления) функций проекта, а также совместно используемые символические константы и макросы выносят в отдельный файл, который подключают к каждому файлу исходного кода. Такие файлы называются **заголовочными.**

В отличие от заголовочных файлов стандартной библиотеки, заголовочные файлы, которые относятся только к вашему проекту, при подключении к файлу исходного кода заключаются в кавычки, а не скобки.

Создадим заголовочный файл, например, **myprint.h** и поместить туда прототипы функций l2r() и r2l(). А в файле main.c следует прописать директиву препроцессора:

#include "myprint.h"

Технология сборки библиотеки

В языке программирования **C** код библиотек представляет собой функции, размещенные в файлах, которые скомпилированы в объектные файлы, а те, в свою очередь, объединены в библиотеки.

У каждой библиотеки должен быть свой заголовочный файл.

При компиляции программы требуется указать имя библиотеки и ее местоположение.

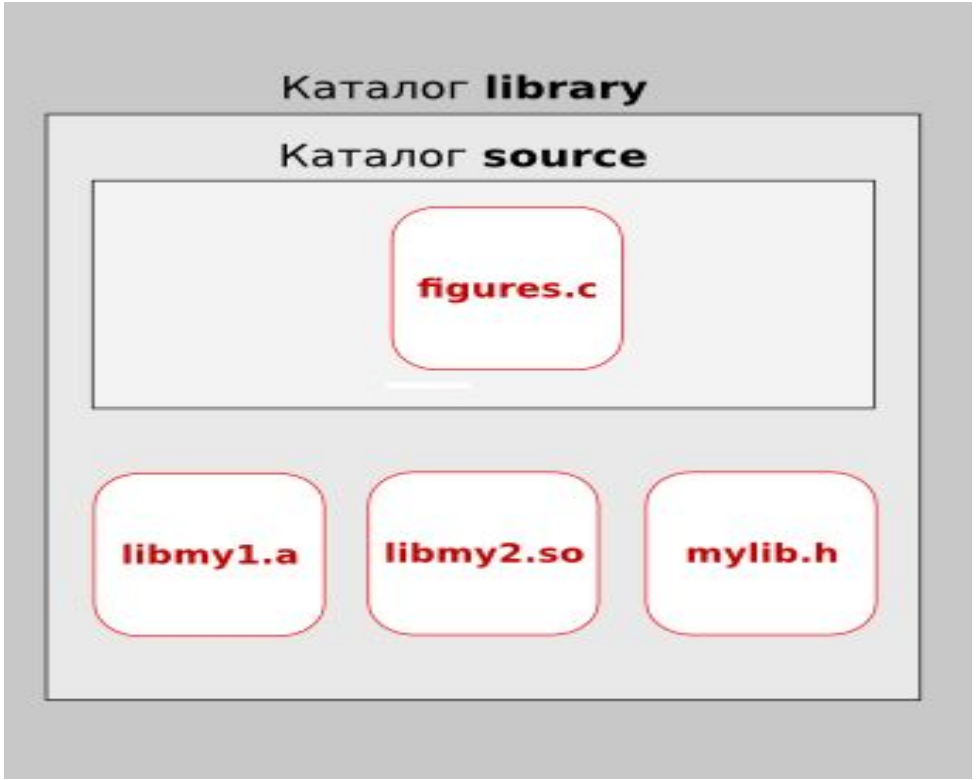
Библиотеки бывают двух видов — **статические** и **динамические**.

Код первых при компиляции полностью входит в состав исполняемого файла, что делает программу легко переносимой.

Код динамических библиотек не входит в исполняемый файл, последний содержит лишь ссылку на библиотеку.

Пример создания библиотеки

В операционных системах GNU/Linux имена файлов библиотек должны иметь префикс "lib", статические библиотеки - расширение *.a, динамические - *.so.



В файле figure.c содержатся две функции — rect() и diagonals().

Первая принимает в качестве аргументов символ и два числа и "рисует" на экране с помощью указанного символа прямоугольник заданной ширины и высоты.

Вторая функция выводит на экране две диагонали квадрата ("рисует" крестик).

Исходный код библиотеки

Файл figures.c

```
#include "stdafx.h"
void rect (char sign, int width, int height) {
int i, j;
for (i=0; i < width; i++) putchar(sign);
putchar('\n');
for (i=0; i < height-2; i++) {
for (j=0; j < width; j++) {
if (j==0 || j==width-1) putchar(sign);
else putchar(' ');} putchar('\n');
}
```

```
for (i=0; i < width; i++) putchar(sign);
putchar('\n');
}
void diagonals (char sign, int width) {
int i, j;
for (i=0; i < width; i++) {
for (j=0; j < width; j++) {
if (i == j || i+j == width-1) putchar(sign);
else putchar(' ');
} putchar('\n');
}
}
```

Заголовочный файл

Файл mylib.h:

```
void rect (char sign, int width, int height);
void diagonals (char sign, int width);
```

лучше сохраним его там, где будут библиотеки

Создание статической библиотеки

1)Перейдем в каталог library (команда cd)

2)Получаем объектные файлы:

gcc -c ./source/*.c

В итоге в каталоге library должно наблюдаться следующие объекты:

figures.o mylib.h source

3)Далее используем утилиту **ar** для создания статической библиотеки:
ar r libmy1.a *.o

Параметр r позволяет вставить файлы в архив, если архива нет, то он создается. Далее указывается имя архива, после чего перечисляются файлы, из которых архив создается.
Объектные файлы нам не нужны, поэтому их можно удалить:
rm *.o

В итоге содержимое каталога library должно выглядеть так:

libmy1.a mylib.h

, где **libmy1.a** — это статическая библиотека.

Файл proba.c

```
#include "stdio.h"
#include "mylib.h"
int  main( )
{  rect('-',25,6);
    printf("\n\n");
rect('+',25,3);
printf("\n\n");
diagonals ('~', 10);
    return 0;
}
```

Компиляция проекта со статической библиотекой:

gcc -o project *.o -L../library -lmy1

Начало команды должно быть понятно: опция -o указывает на то, что компилируется исполняемый файл project из объектных файлов. Здесь опция -L указывает на адрес каталога, где находится библиотека, он и следует сразу за ней. После опции -l записывается имя библиотеки, при этом префикс lib и суффикс (неважно .a или .so) усекаются. Обратите внимание, что после данных опций пробел не ставится.

Создание динамической библиотеки

Объектные файлы для динамической библиотеки компилируются **особым образом**. Они должны содержать так называемый позиционно-независимый код (position independent code).

Компиляция объектных файлов для динамической библиотеки должна выполняться с опцией **-fPIC** компилятора gcc:

```
gcc -c -fPIC source/*.c
```

В отличие от статической библиотеки динамическую создают при помощи gcc указав опцию **-shared**:

```
gcc -shared -o libmy2.so *.o
```

Использованные объектные файлы можно удалить:

```
rm *.o
```

В итоге содержимое каталога library:

```
libmy1.a libmy2.so mylib.h source
```

Компиляция проекта с динамической библиотекой

Теперь удалим исполняемый файл и получим его уже связанным с динамической библиотекой. Команда компиляции с динамической библиотекой выглядит так:

gcc -o project *.o -L../library -lmy2 -Wl,-rpath,../library/

Здесь в отличии от команды компиляции со статической библиотеки добавлены опции для
линковщика: -Wl,-rpath,../library/. -Wl - это обращение к линковщику, -rpath - опция линковщика, ../library/ - значение опции.

Работа с файлами

Файлы представляют собой именованные области внешней (дисковой) памяти, с которыми программы могут обмениваться информацией.

Файлы предназначены только для хранения информации, а обработка этой информации осуществляется программами.

Работа с файлами складывается из трех шагов:

1. ***Файл открывается.*** Это означает, что программа "захватывает" заданный по имени файл, сообщает ОС, что далее она будет с ним работать.
2. ***Ведется работа с файлом***. Из него данные либо считываются, либо в него записываются.
3. ***Файл закрывается.*** После этой операции он снова доступен другим программам для обработки.

Файлы позволяют пользователю считывать большие объемы данных непосредственно с диска, не вводя их с клавиатуры. Существуют два основных типа файлов: **текстовые и двоичные**.

Текстовыми называются файлы, состоящие из любых символов. Они организуются по строкам, каждая из которых заканчивается символом «*конца строки*». Конец самого файла обозначается символом «*конца файла*».

Для работы с файлами в программах используется специальный тип данных – структура FILE, предназначенная для хранения атрибутов файлов (указатель (адрес) текущей позиции файла, признак конца файла, и др.).

В программе описывается переменная - указатель типа FILE, которая будет представителем данного файла:

```
FILE *f;
```

где *f – переменная-указатель на файл.

1) Открытие файла

fl=fopen("путь к файлу","режим работы файла");

Параметр "путь к файлу" указывает размещение файла на диске. Он обязательно содержит имя файла и может содержать путь к нему.
Параметр "режим работы файла" показывает, как будет использоваться файл:
"w" – для записи данных (вывод);
"r" – для чтения данных (ввод);
"a" – для добавления данных к существующим записям.

Пример открытия файла для записи:

```
FILE * fo;
   fo = fopen("test.txt", "w");
```

Можно задать и полный путь к файлу, например:

```
   fo = fopen("c:\tmp\test.txt", "w");
```

Функция fopen() возвращает значение указателя на структуру типа файл. Если файл по каким-либо причинам не открывается, функция fopen() возвращает значение NULL.

Одновременно с открытием файла можно проверять, успешно ли это сделано:

```
if( (fo=fopen("c:\tmp\test.txt","w")) == 0 )
{
    . . . // ошибка!
  }
```

Если файл открывается в режиме добавления данных "a", то указатель текущей позиции устанавливается на конец файла.

В C++ файл можно открыть для чтения и/или записи в текстовом или бинарном (двоичном) режиме. Поэтому можно указать дополнительные условия режима открытия файла:

"b" – двоичный поток;

"t" – текстовый поток;

"+" – обновление файла.

2) Обработка открытого файла

Для ввода/вывода в C++ используются следующие функции:

Чтение (ввод)	Запись (вывод)
fgetc()	fputc()
fscanf()	fprintf()
fgets()	fputs()
fread()	fwrite()

При каждой операции ввода/вывода указатель текущей позиции файла смещается на одну позицию в сторону конца файла.

Работа с текстовыми файлами

Файлы бывают текстовые (в которых можно записывать только буквы, цифры) и двоичные (в которых могут храниться любые символы из таблицы). В текстовых файлах символы конца строки 0x13 (возврат каретки, CR) и 0x10 (перевод строки LF) преобразуются при вводе в одиночный символ перевода строки (при выводе выполняется обратное преобразованиеПри обнаружении в текстовом файле символа с кодом 26 (0x26), т.е. признака конца файла, чтение файла в текстовом режиме заканчивается.

Рассмотрим пример вывода в файл значения переменной:

```
  #include "stdio.h"
int _tmain(int argc, _TCHAR* argv[])
{int n=10;
  char st[25] = "значение переменной n=";
  FILE *fo;
  fo = fopen("test.txt","wt");
  fprintf( fo, "Вывод: %s %d", st, n );
  fclose(fo);
  return 0;}
```

Проверка признака конца файла

Так как при каждой операции ввода/вывода происходит перемещение указателя в файле, в какой-то момент указатель достигает конца файла.

Функция feof() проверяет состояние индикатора конца файла и возвращает значение 0, если конец файла не был достигнут, или значение, отличное от нуля, если был достигнут конец файла.

Пример вызова функции в команде if:

if (! feof(fo))…

Функции ввода/вывода

1)Посимвольный ввод/вывод:fgetc()/fputc()

Функция getc() выбирает из файла очередной символ; ей нужно только знать указатель на файл, например,

char S=fgetc(f);

Где f –файловая переменная;

Функция putc() заносит значение символа S в файл, на который указывает указатель f. Формат вызова функции:

fputc(S,f);

Пример1. Текст из файла my_char.txt выводится на экран.

```
#include "stdafx.h"
int main()
{FILE *ptr;    //описание указателя на файл
char ch;
ptr=fopen("my_char.txt","r");//открытие файла
ch=fgetc(ptr);  //чтение первого символа
while (!feof(ptr))  //цикл до конца файла
{ printf("%c",ch);    //вывод символа на экран
ch=fgetc(ptr);      //чтение  следующего символа
} fclose(ptr);      //закрытие файла
return 0;}
```

2) Построковый ввод/вывод:fgets()/fputs()
Функция fputs( ) записывает строку символов в файл. Она отличается от функции puts( ) тем, что в качестве второго параметра должна быть указана переменная файлового типа.
Например:

```
fputs("Example", f);
```

Функция fgets( ) читает строку символов из файла.

```
fgets(string, n, f);
```

Пример2. Создать текстовый файл на диске.

```
#include "stdio.h"
#include "string.h"
int _tmain(int argc, _TCHAR* argv[])
{int n=10;
  char st[25];
  FILE *fo;
  fo = fopen("test.txt","wt");
  while(strcmp(st,"end")!=0)
  {gets(st);
  if(strcmp(st,"end")!=0) fputs(st,fo);
  }printf("конец создания файла");
  fclose(fo);
  return 0;}
```

2) форматированный ввод/вывод:fscanf()/fprintf()
Чтение из файла выполняет функция fscanf():

fscanf(f,”строка формата”,список адресов перем-х);

где f –файловая переменная;
Запись в файл осуществляет функция fprintf():

fprintf(f,"строка формата",список переменных);

С помощью этих функция можно вводить и выводить значения переменных любого стандартного типа

Пример 3. Записать массив в файл:

```
#include "stdio.h "
int main()
{ int i;
int array[]={5,15,135,3}; //описание и инициализация
FILE *out;                //описание указателя на файл
out=fopen("num_arr.txt","w"); //открытие файла
for(i=0;i<4;i++)
fprintf(out,"%5d",array[i]);//запись в файл элемента
fclose(out);              //закрытие файла
return 0;
}
```

Функция remove( ) удаляет файл с заданным именем:

```
remove("my_file.txt");
```

Здесь **"my_file.txt"** – имя удаляемого файла. При успешном завершении возвращается нуль, в противном случае возвращается ненулевое значение.

Функция rewind( ) устанавливает указатель текущей позиции в начало файла и имеет следующий прототип:

```
rewind(fp);
```

Где fp - файловая переменная

Функция rename( ) переименовывает существующий файл :

```
rename("old_file.txt", "new_file.txt");
```

Пример 4. Файл содержит текст в котором есть цифры. Удалить цифры из файла.

```
#include "stdio.h"
#include <string.h>
int main()
{ FILE *f, *r;          // Указатели на файлы
  char ch, pr[65];
  f=fopen("FIL1.txt","r");
  r=fopen("FIL2.txt","w");
```

```
while (!feof(f))  //Цикл пока не конец файла
  { ch=fgetc(f);  // Чтение символа ch из файла
    if (!(ch >='0'&&ch<='9')&& !feof(f))
   /*Если прочитанный символ не цифра и не
    конец файла*/
    fputc(ch,r); // Запись в файл r символа ch
  }
    fclose(f);fclose(r);
 remove("FIL1.txt");
 rename("FIL2.txt","FIL1.txt");
return 0;
}
```

Обработка бинарных файлов

Если файл открыт в бинарном режиме, его можно записывать или считывать побайтно.

Бинарный файл – это файл произвольного доступа.

Функция fseek() позволяет переходить к любой позиции в файле, обеспечивая возможность произвольного доступа.

Функция fseek( ) имеет формат:

```
fseek(fp, count, access);
```

где fp - указатель на файл;
count - номер байта относительно заданной позиции, начиная с которого будут затем выполняться операции;
access - способ задания начальной позиции.
Переменная access может принимать следующие значения:
0 - начальная позиция задана в начале файла;
1 - начальная позиция считается текущей;
2 - начальная позиция задана в конце файла.
При успешном завершении возвращается нуль, при ошибке - ненулевое значение.

Функции ввода-вывода для бинарных файлов

Функция fread( ) предназначена для чтения блоков данных из потока:

fread(ptr, size, n, fp);

Она читает n элементов данных, длиной size байт каждый, из файла fp в переменную с адресом ptr. Общее число прочитанных байтов равно произведению n*size.

При успешном завершении функция fread( ) возвращает число прочитанных элементов данных, при ошибке - 0.

Функция fwrite( ) предназначена для записи в файл блоков данных:

fwrite(ptr, size, n, fp);

Она добавляет n элементов данных, длиной size байт каждый, в заданный выходной файл fp. Данные записываются из переменной с адресом ptr.

При успешном завершении операции функция fwrite( ) возвращает число записанных элементов данных, при ошибке - неверное число элементов данных.

Пример 5. Составим программу создания нового файла с информацией о городах: код, название, численность жителей.

```
#include "stdio.h”
typedef struct city
    { int kod;
      char name[10];
      long  c; } town;
town t;
int main()
{setlocale(0,"Rus");
 FILE *f; char ch;
```

```
 f=fopen("file1.dat","wb"); //открытие для записи
 printf("\n Ввод информации о городе ");
 do
{ printf("\nКод: "); scanf("%d", &t.kod);
  printf("\nназвание: "); scanf("%s", t.name);
  printf("\nкол-во жителей: "); scanf("%ld", &t.c);
  fwrite(&t, sizeof(t), 1, f); //запись в файл стр-ры t
   printf("\n END Закончить? ( y/n): "); scanf("\n");
   ch=getch();
 }
 while (ch != 'y');
 fclose(f);
}
```

Пример 6. Вывести на экран список городов, количество жителей в которых превышает миллион человек.

```
#include "stdio.h "
 struct town
    { int kod;
      char name[10];
      long  c; };
town t;
int main()
{
  FILE *f;
```

```
f=fopen("file1.dat","rb"); //открытие файла для чтения
 fread(&t, sizeof(t), 1, f); //чтение из файла t
 while (!feof(f))
 { if(t.c>1000000)
  printf("\n   %10s  количество жителей: %ld",
   t.name, t.c);
  fread(&t, sizeof(t), 1, f);
 }
  fclose(f);
}
```

Редактирование бинарного файла

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
struct tov {
      char name[10];
      float c;
       int kol;} t1;
void input(FILE *);
void print(FILE *);
void app(FILE *);
void find(FILE *);
```

```
int main()
{ char c;   FILE *tf;
  while (1)
  {
     printf("\n");
    puts("  1 sozdanie");
    puts("  2 procmotr");
    puts("  3 dobavlenie");
    puts("  4 poisk i
redaktirovanie");
    puts("  0 exit");
    c=getch();
```

```
switch(c)
   { case '1': tf=fopen("file1.dat","wb");input(tf);
break;
     case '2':  tf=fopen("file1.dat","rb"); print(tf);
break;
     case '3':   tf=fopen("file1.dat","ab");app(tf);
break;
     case '4':   tf=fopen("file1.dat","rb+");find(tf);
break;
     case '0': return 0;
     default : puts(" ne verno");
    }
  }}
```

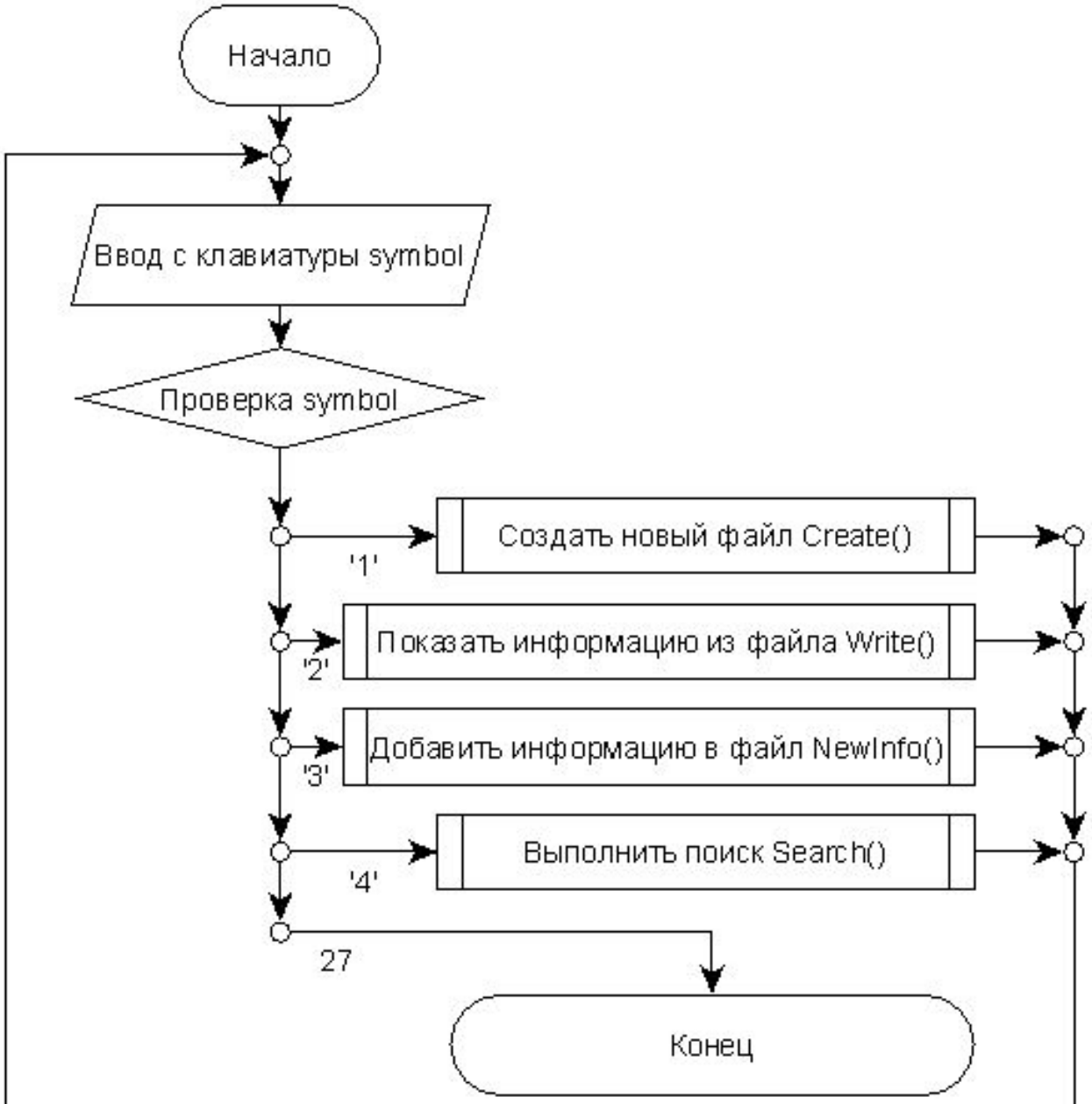
```
void input(FILE *tf)
{ char ch;
printf("\n vvod\n");
  do
  { printf("\n name: "); scanf("%s", t1.name);
    printf(" cena: "); scanf("%f", &t1.c);
    printf(" kol-vo: "); scanf("%d", &t1.kol);
    fwrite(&t1, sizeof(t1), 1, tf);
    printf("\n konec vvoda?  y/n  ");
    ch=getch();
  }
  while (ch != 'y');
  fclose(tf);
}
```

```
void print(FILE *tf)
{ int i;

  i=1;
  fread(&t1, sizeof(t1), 1, tf);
  while (!feof(tf))
  { printf("\n %3d tovar:%s cena:%6.2f  kol:
%4d", i, t1.name, t1.c, t1.kol);
    fread(&t1, sizeof(t1), 1, tf);
    i++;
  } fclose(tf);
  getch();}
```

```
void app(FILE *tf)
{ char ch;
  printf("\n  vvod \n");
  do
  {  printf("\n name: "); scanf("%s", t1.name);
    printf(" cena: "); scanf("%f", &t1.c);
    printf(" kol-vo: "); scanf("%d", &t1.kol);
    fwrite(&t1, sizeof(t1), 1, tf);
    printf(" konec vvoda?  y/n ");
    ch=getch();
  }
  while (ch != 'y');
  fclose(tf);  }
```

```
void find(FILE *tf)
{ char c,tov[10];
  long int i;
  puts(" vvod name: ");
  gets(tov);
  fread(&t1, sizeof(t1), 1, tf);
  while (!feof(tf))
  {  if (strcmp(t1.name, tov)==0)
     {   printf(" name  %10s cena %6.2f  kol  %d",
t1.name, t1.c, t1.kol);
      printf("\n zamena? y/n ");
      c=getch();
      if (c=='y')
```

```
        {  printf("\n kol: "); scanf("%d", &t1.kol);
        printf("\n cena: "); scanf("%f", &t1.c);
        i=sizeof(t1);
        fseek(tf, -i, 1);
        fwrite(&t1, sizeof(t1), 1, tf);
         fseek(tf,0,2);
      }
      }
     fread(&t1, sizeof(t1), 1, tf);

  }
  fclose(tf);
}
```


Начало
Ввод с клавиатуры symbol
Проверка symbol
'1'
Создать новый файл Create()
'2'
Показать информацию из файла Write()
'3'
Добавить информацию в файл NewInfo()
'4'
Выполнить поиск Search()
27
Конец