Язык Си в примерах/Сортировка

Программа на языке Си — это один или несколько текстовых файлов, которые также называются исходными.

Исполнить исходные файлы нельзя, их необходимо скомпилировать, т.е. создать исполняемый файл, содержащий в себе инструкции процессора и пригодный для запуска на компьютере.

Процесс преобразования исходных файлов в исполняемый называется компиляцией. Если ваша программа состоит из одного исходного файла hello.c, то для его компиляции компилятором GNU С достаточно выполнить команду:

bash$ gcc hello.c -o hello

В результате получится файл hello, имя которого мы указали в опции -o. Этот файл является исполняемым и его можно запускать (execute) при помощи команды:

bash$ ./hello

Пара символов ./ перед hello означает "искать исполняемый файл hello в текущей директории".

Строчка

bash$ gcc xxx.c yyy.c -o zzz -I./common -I.. -lm

соответствует команде: "скомпилировать файлы xxx.c yyy.c в программу zzz; заголовочные файлы находятся в директориях ./common и ..; подключить библиотеку libm"

Библиотека libm (подключаемая с помощью опции -lm) содержит откомпилированные математические функции, которые объявляются в заголовочном файле math.h. Если вы используйте функции из этой библиотеки (такие как log, sin, cos, exp), то не забывайте подключать её при компиляции.

Подробную информацию об опциях компилятора gcc можно получить, если набрать

bash$ man gcc

или

bash$ info gcc

Первая программа, которую мы рассмотрим, — это ***«Hello World»*** — программа, которая выведет на экран строчку «Hello, World!» и закончит своё выполнение.

#include <stdio.h>

int main (void) {

printf ("Hello, World!\n");

return 0;

}

Вот текст программы**, складывающей два введенных целых числа:**

#include <stdio.h>

int main () {

int a, b;

printf ("Введите два числа: ");

scanf ("%d%d", &a, &b);

printf ("%d\n", a + b);

return 0;

}

Задача: написать программу, которая находит **максимум введенных целых чисе**л. На вход поступает количество чисел, а затем сами числа.

Вот решение этой задачи:

/\* Программа max.c "Максимум чисел" \*/

#include <stdio.h>

int main () {

int i, n, a, max;

printf ("Введите количество чисел: ");

scanf ("%d", &n);

printf ("Введите %d чисел: ", n);

scanf ("%d", &max);

for(i = 1; i < n ; i++) {

scanf ("%d", &a);

if(a > max)

max = a;

}

printf ("%d", max);

return 0;

}

Задача: вывести на экран компьютера **таблицу умножения размера n x n**, где n вводит пользователь с клавиатуры.

Решение задачи:

/\* Программа table.c "Таблица умножения" \*/

#include <stdio.h>

int main()

{

int i, j, n;

printf ("Введите n: ");

scanf ("%d", &n);

for(i = 1; i <= n ; i++) {

for(j = 1; j <= n ; j++) {

printf ("%5d", i \* j);

}

printf("\n");

}

return 0;

}

В приведенной ниже программе определена **функция factorial**, вычисляющая факториал.

#include <stdio.h>

int factorial(int x) {

if( x == 0 ) return 1;

return x \* factorial (x - 1);

}

void main() {

int n;

while( scanf("%d", &n) == 1)

printf("%d\n", factorial (n));

}

/\*

Программа 2: **степень числа -- оптимизированная рекурсия.**

\*/

double power(double x, long n)

{

double tmp;

if(n == 0) return 1;

if(n < 0) return power ( 1 / x, -n);

if(n % 2) return x \* power (x, n - 1);

return power(x \* x, n / 2);

}

/\*

Программа 3: **cтепень числа -- оптимальный алгоритм без рекурсии.**

\*/

double power(double x, long n)

{

double a = 1;

while(n) {

if(n % 2) {

a \*= x;

n--;

}

else{

x \*= x;

n /= 2;

}

}

return a;

}

**/\* Универсальная функция вычисления корня уравнения f(x) = 0 \*/**

#include <stdio.h>

#include <math.h>

#define EPS 1e-16

double root(double l, double r, double (\*f)(double)) {

double c;

while( r - l > EPS ) {

c = ( l + r ) / 2;

if( f(c) \* f(r) < 0 )

l = c;

else

r = c;

}

return l;

}

double f1(double x) {return cos(x) - 3 \* x; }

double f2(double x) {return exp(x) - x - 2; }

int main() {

printf("root1 = %lf\n", root(0, 2, f1)); // вычисляем и выводим корень уравнения f1(x) = 0

printf("root2 = %lf\n", root(0, 2, f2)); // вычисляем и выводим корень уравнения f2(x) = 0

return 0;

}

Вывод программы выглядит следующим образом:

root1 = 0.316751

root2 = 1.146193

***Пример использования функции qsort из stdlib***

***Вход: В первой строчке дано N, а затем следует N строчек вида ИМЯ ТЕЛЕФОН***

***Выход: Выведите телефонный справочник в алфавитном порядке и в порядке возрастания телефонных номеров***.

#include <stdio.h>

#include <stdlib.h>

typedef struct person\_t {

char \* name;

int phone;

} person\_t;

int namecmp(const void\* a, const void\* b);

int phonecmp(const void\* a, const void\* b);

int main(void)

{

char name[1000];

person\_t \*base;

int n, i, phone;

scanf("%d", &n);

base = malloc(n \* sizeof(person\_t));

/\* Считываем телефонную базу \*/

for(i = 0 ; i < n ; i++)

{

scanf("%s%d", name, &phone);

/\* Выделяем память под массив base[i].name,

учитывая символ конца строки \*/

base[i].name = malloc((strlen(name)+1) \* sizeof(char));

strcpy(base[i].name, name);

base[i].phone = phone;

}

qsort(base, n, sizeof(person\_t), namecmp);

/\* Напечатаем в алфавитном порядке \*/

for(i = 0 ; i < n ; i++)

{

printf("%20s %d\n", base[i].name, base[i].phone);

}

qsort(base, n, sizeof(person\_t), phonecmp);

/\* Напечатаем в порядке возрастания номеров\*/

for(i = 0 ; i < n ; i++)

{

printf("%20s %d\n", base[i].name, base[i].phone);

}

/\* Освобождаем выделенную память \*/

for(i = 0 ; i < n ; i++)

{

free(base[i].name);

}

free(base);

return 0;

}

int namecmp(const void\* a, const void\* b)

{

person\_t \*pa = a;

person\_t \*pb = b;

return strcmp(pa->name, pb->name);

}

int phonecmp(const void\* a, const void\* b)

{

person\_t \*pa = a;

person\_t \*pb = b;

return pa->phone - pb->phone;

}

***Задача «сортировки» (упорядочения)*** — одна из первых интересных и сложных задач теории алгоритмов. Общие принципы освещает статья «Алгоритмы сортировки» в журнале «Потенциал»; здесь же мы рассматриваем способы упорядочения посредством языка Си.Содержание [убрать]

1 Метод «пузырька»

2 Функция qsort из библиотеки stdlib

3 Динамическое выделение памяти

4 Программа упорядочения строк в алфавитном порядке

5 Примечания

**Метод «пузырька»**

Один из простейших алгоритмов решения — «метод пузырька».

#include<stdio.h>

#define N 1000

int main() {

int n, i, j;

int a[N];

// считываем количество чисел n

scanf("%d", &n);

// считываем n чисел

for(i = 0 ; i < n; i++) {

scanf("%d", &a[i]);

}

for(i = 0 ; i < n ; i++) {

// сравниваем два соседних элемента.

for(j = 0 ; j < n - i - 1 ; j++) {

if(a[j] > a[j+1]) {

// если они идут в неправильном порядке, то

// меняем их местами.

int tmp = a[j]; a[j] = a[j+1] ; a[j+1] = tmp;

}

}

}

}

**Функция qsort из библиотеки stdlib**

Два оператора for, в которых происходит сортировка, можно заменить на одну строку:

qsort(a, n, sizeof(int), cmp );

Это функция, описанная в стандартной библиотеке ANSI C и объявлена в заголовочном файле stdlib.h.

Поэтому в начале программы нужно добавить

#include <stdlib.h>

Функцией qsort можно упорядочивать объекты любой природы. По сути, она предназначена упорядочивать множества блоков байтов равной длины. Второй аргумент функции — это число таких блоков, третий аргумент — длина каждого блока. Первый аргумент — это адрес, где находится начало первого блока (предполагается, что блоки в памяти расположены друг за другом подряд).

Четвёртый аргумент функции qsort — это имя функции, которая умеет сравнивать два элемента массива. В нашем случае это

int cmp(const void \*a, const void \*b) {

return \*(int\*)a - \*(int\*)b;

}

В силу указанной универсальности функции сортировки, функция сравнения получает в качества аргумента адреса двух блоков, которые нужно сравнить и возвращает 1, 0 или -1:

положительное значение, если a > b

0, если a == b

отрицательное значение, если a < b

Поскольку у нас блоки байт -- это целые числа (в 32-битной архитектуре это четырёхбайтовые блоки), то необходимо привести данные указатели типа (const void\*) к типу (int \*) и осуществляется это с помощью дописывания перед указателем выражения «(const int\*)». Затем нужно получить значение переменной типа int, которая лежит по этому адресу. Это делается с помощью дописывания с переди звездочки.

Таким образом, мы получили следующую программу

#include <stdio.h>

#include <stdlib.h>

#define N 1000

int cmp(const void \*a, const void \*b) {

return \*(int\*)a - \*(int\*)b;

}

int main() {

int n, i,j;

int a[N];

scanf("%d", &n);

for(i = 0 ; i < n; i++) { // ЧИТАЕМ ВХОД

scanf("%d", &a[i]);

}

qsort(a, n, sizeof(int), cmp ); // СОРТИРУЕМ

for(i = 0 ; i < n; i++) { // ВЫВОДИМ РЕЗУЛЬТАТ

printf("%d ", a[i]);

}

return 0;

}

Динамическое выделение памяти

Ниже приведена программа, где память под массив выделяется динамически:

#include <stdio.h>

#include <stdlib.h>

#include <malloc.h>

#define N 1000

int cmp(const void \*a, const void \*b) {

return \*(int\*)a - \*(int\*)b;

}

int main() {

int n, i;

int \*a;

scanf("%d", &n);

a = (int\*) malloc(sizeof(int)\*n);

for(i = 0 ; i < n; i++) {

scanf("%d", &a[i]);

}

qsort(a, n, sizeof(int), cmp );

for(i = 0 ; i < n; i++) {

printf("%d ", a[i]);

}

free(a);

return 0;

}

**Функция malloc** (от англ. memory allocation --- выделение памяти) делает запрос к ядру операционной системы по выделению заданного количества байт. Единственный аргумент этой функции — число байт, которое вам нужно. В качестве результата функция возвращает указатель на начало выделенной памяти. Указатель на начало выделенной памяти &mbsah — это адрес ячейки памяти, начиная с которого идут N байт, которые вы можете использовать под любые свои нужды. Всю память, которая была выделена с помощью функции malloc, нужно освобождать с помощью функции free. Аргумент функции free — это указатель на начало выделенной когда-то памяти.

[править]

Программа упорядочения строк в алфавитном порядке

#include <stdlib.h>

#include <string.h>

#include <stdio.h>

#define N 100

#define M 30

int main(int argc, char\* argv[]) {

char a[N][M];

int n,i;

scanf("%d",&n);

for (i=0;i<n;i++)

scanf("%s", &a[i]);

qsort(a, n, sizeof(char[M]), (int (\*)(const void \*,const void \*)) strcmp );

for (i=0;i<n;i++)

printf("%s\n",a[i]);

return 0;

}

Обратите внимание на сложное приведение типов.

**Функция strcmp**, объявленная в файле string.h имеет следующий прототип

int strcmp(const char\*, const char\*);

То есть функция получает два аргумента -- указатели на кусочки памяти, где хранятся элементы типа char, то есть два массива символов, которые могут быть изменены внутри функции strcmp (запрет на изменение задается с помощью модификатора const)[1].

В то же время в качестве четвертого элемента функция qsort хотела бы иметь функцию типа

int cmp(const void\*, const void\*);

В языке Си можно осуществлять приведение типов являющихся типами функции. В данном примере тип

int (\*)(const char\*, const char\*); // функция, получающая два элемента типа 'const char \*' и возвращающая элемент типа 'int'

приводится к типу

int (\*)(const void\*, const void\*); // функция, получающая два элемента типа 'const void \*' и возвращающая элемент типа 'int'

Примечания

^ Функция strcmp в соответствии с описанием, выдаваемой командой man 3 strcmp, осуществляет сравнение двух строк

и определяет какая из двух строк идёт первой в алфавитном порядке (стравнивает две строки в лексографическом порядке), а именно, она возвращает 1, если первая строка больше второй (идёт после второй в алфавитном порядке), 0 – если они совпадают, и -1 – если первая меньше второй.