Основы программирования

Ст. преп.

каф. ПОВТ

Масленников Алексей Александрович

Лекция № 5

Функции и структура программы

Функции и структура программы

Модульность в языках программирования – это принцип, согласно которому программное средство (ПС) – программа, библиотека, веб-приложение и др. разделяется на отдельные сущности, называемые модулями. Модульность позволяет упростить задачи проектирования ПС и распределения процесса разработки ПС между группами разработчиков, а также позволяет реализовать методологию *повторного использования кода*.

Функции и структура программы

- В языке С модульность поддерживается:
  - функциями,
  - препроцессоными командами,
  - многофайловой структурой программы и
  - заголовочными файлами.

Функции и структура программы

Текст программа на языке С может весь размещаться в одном файле и такая программа будет иметь *однофайловую* структуру, либо текст программы может быть размещен в нескольких файлах и тогда программа будет *многофайловой*.

Функции и структура программы

Текст программа на языке С может весь размещаться в одном файле и такая программа будет иметь *однофайловую* структуру, либо текст программы может быть размещен в нескольких файлах и тогда программа будет *многофайловой*.

Функции и структура программы

С помощью *функций* большие вычислительные задачи разбивают на более мелкие. Это позволяет инкапсулировать («упрятать» в оболочку) детали *реализации* некоторой функциональности и предоставить пользователям («клиентам») формат обращения к этой функциональности (*интерфейс*). В результате программа в целом становится более ясной и в нее проще вносить изменения.

Пример. Функция-парсер atoi_(), преобразующая символьное изображение числа, записанное в строке, в само число, которая затем вызывается в основной программе. В этом примере программа распределена по двум файлам: main.c, в котором находится основная программа и описывается интерфейс функции atoi_() и atoi_.c, который содержит реализацию этой функции.

Функции и структура программы

```
// файл main.c
#include <stdio.h>
int atoi_(char[]); // описание интерфейса функции
int main() {
  int n = atoi_("  -347ab");
  printf("n = %d\n", n);
  return 0;
}
```

```
// файл atoi_.c – описание реализации функции
#include <ctype.h>
int atoi_(char s[]) {
  int i, n, sign;
  for (i = 0; isspace(s[i]); i++) ;
  sign = (s[i] == '-') ? -1 : 1;
  if (s[i] == '+' || s[i] == '-') i++;
  for (n = 0; isdigit(s[i]); i++)
    n = 10 * n + (s[i] - '0');
  return sign * n;
}
```

Пример. Разработаем программу, позволяющую находить с заданной точностью корень нелинейного уравнения f(x)=0 методом деления отрезка пополам. Пусть заданы: функция f(x), отрезок [a, b], на котором предположительно находится корень x0; точность определения нуля функции – epsy и точность определения корня – epsx.

Суть метода состоит в вычислении значения функции посередине текущего отрезка, сравнении полученного значения с *epsy* и переносе левой или правой границы в середину, если не достигнута требуемая точность.

Далее приведен текст однофайловой программы, которая отыскивает корень функции cos(*x*) на отрезке [0, 3]:

Функции и структура программы

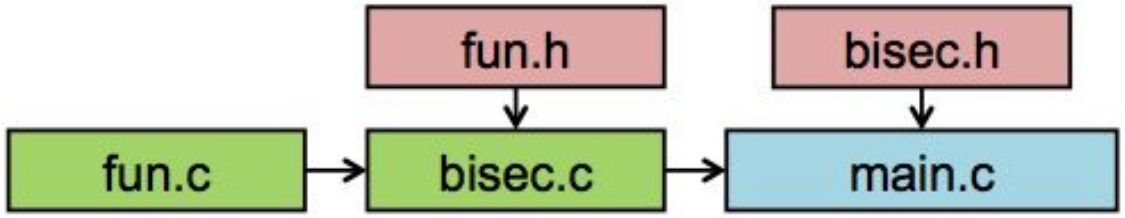
```
double fun(double x) {
  return cos(x);
}
double bisec(double a, double b, double epsx,
             double epsy) {
  double y1, y2, x;
  if(fabs(y1=fun(a)) < epsy) return a;
  if(fabs(y2=fun(b)) < epsy) return b;
  if(y1*y2 > 0.0) {
    printf("bisec: no root...\n");
    return 1.e10;
  }
  while(1) {
    x = (a+b)/2.0;
    if(fabs(y1=fun(x)) < epsy) return x;
    if(y1*y2 > 0.0) b = x; else a = x;
    if(fabs(a-b) < epsx) return x;
  }
}
int main() {
  printf("result=%f\n",bisec(0., 3., 1e-10, 1e-10));
  return 0;
}
```

Функции и структура программы

При разработке больших по объему кода программ правильнее применять модульный подход и разбивать программу на отдельные модули (файлы).

При этом рекомендуется создавать по два модуля на каждую функцию (кроме main): модуль реализации (файл *.c) и модуль интерфейса (файл *.h).

С использованием этих рекомендаций представим программу поиска к



Функции и структура программы

```
// файл main.c
#include <stdio.h>
#include "bisec.h"
int main() {
  printf("result=%f\n",bisec(0., 3., 1e-10, 1e-10));
  return 0;
}
```

```
// файл fun.h
double fun(double);
```

```
// файл fun.c
#include <math.h>
double fun(double x) {
  return cos(x);
}
```

```
// файл bisec.h
double bisec(double, double, double, double);
```

```
// файл bisec.c
#include <stdio.h>
#include <math.h>
#include "fun.h"
double bisec(double a, double b, double epsx,
             double epsy) {
  double y1, y2, x;
  if(fabs(y1=fun(a)) < epsy) return a;
  if(fabs(y2=fun(b)) < epsy) return b;
  if(y1*y2 > 0.0) {
    printf("bisec: no root...\n");
    return 1.e10;
  }
  while(1) {
    x = (a+b)/2.0;
    if(fabs(y1=fun(x)) < epsy) return x;
    if(y1*y2 > 0.0) b = x; else a = x;
    if(fabs(a-b) < epsx) return x;
  }
}
```

Определение функции и структура программы

Для того, чтобы использовать функцию, ее необходимо определить, т.е. описать её интерфейс (объяснить, как функцией можно воспользоваться) и привести программный код (тело функции), раскрывающий, как функция работает (записать реализацию функции на языке программирования).

Функции и структура программы

Определение функции имеет следующую форму:

```
тип_возвращ_знач имя_функции(список_объявлений_арг) {
  объявления и операторы
}
```

Различные части этого определения могут отсутствовать, но обязательными являются: имя_функции, пара круглых скобок и пара фигурных скобок, т.е. «минимальная» функция определяется так:

```
fun(){
}
```

Определение функции и структура программы

Если при объявлении функции не указан тип возвращаемого значения, то по умолчанию подразумевается тип int. Отсутствие списка объявлений аргументов означает, что функция не использует аргументы.

Функции обмениваются данными посредством передачи аргументов и возвращения значений, а также через внешние переменные.

Функции могут следовать друг за другом в файле исходного кода в любом порядке, и текст программы можно разбивать на любое количество файлов, но при этом запрещается разбивать текст функции между файлами. Функция не может быть также определена внутри другой функции.

Определение функции и структура программы

В результате своей работы функция может возвратить в вызывающую ее функцию результат – некоторое значение, тип которого объявлен перед именем функции. Для этого в теле функции должен присутствовать хотя бы один оператор возврата вида:

Вызывающая функция может игнорировать (т.е. не использовать) возвращаемое значение

Существует еще одна форма оператора возврата:

```
return;
```

В этом случае в вызывающую функцию ничего не передается, а при определении функции в качестве типа возвращаемого значения указывается void. В теле функции может отсутствовать оператор возврата return. В этом случае возврат из функции происходит при достижении конца тела функции (закрывающей скобки }).

Определение функции и структура программы

Аргументы функции передаются в нее по значению, т.е. при вызове функции значения аргументов присваиваются временным (внутренним) переменным и поэтому все действия функции со своими аргументами никак не отражаются на переменных, переданных функции при вызове.

Если же требуется, чтобы функция оперировала с переменными, которые ей были переданы в качестве аргументов, то нужно использовать указатели (адреса переменных-аргументов).

Определение функции и структура программы

Программа может использовать любое количество функций при одном условии: в программе обязательно должна присутствовать в точности одна функция с именем main («главная» функция), так как запуск программы на выполнение операционной системой производится всегда через эту функцию (т.е. main начинает выполняться первой).

Определение функции и структура программы

Существует две формы main:

- без аргументов – main() и
- с аргументами – main(int, char**).

Вторая форма main позволяет при вызове программы из командной строки передать в нее произвольное количество строковых аргументов.

Следующая программа, которая названа factorial, позволяет вычислить факториалы нескольких чисел, которые задаются как

```
int fact(int n) {
  return n ? n * fact(n-1) : 1;
}
int main(int argc, char **argv) {
  int n;
  while(--argc > 0) {
    n = atoi(*++argv);
    printf("%2d! = %10d\n", n, fact(n));
  }
  return 0;
}
```

Вызов программы будет выглядеть так:
factorial 4 6 8 10

Рекурсия – процесс повторения чего-либо самоподобным способом. Например, вложенные отражения, производимые двумя точно параллельными друг другу зеркалами, являются одной из форм бесконечной рекурсии.

Наиболее общее применение рекурсия находит в математике и информатике. Здесь она является методом определения функций, при котором определяемая функция применена (вызвана) в теле своего же собственного определения.

Определение функции и структура программы

При этом бесконечный набор случаев (значений функции) описывается с помощью конечного выражения, которое для некоторых случаев может ссылаться на другие случаи, если при этом не возникает циклов или бесконечной цепи ссылок.

Фактически это способ определения множества объектов через самого себя с использованием ранее заданных частных определений.

Определение функции и структура программы

Примеры рекурсии в математике:

- факториал целого неотрицательного числа n определяется так: $n! = n * (n-1)!$, $0! = 1$;
- числа Фибоначчи: $F_n = F_{n-1} + F_{n-2}$, $F_1 = F_2 = 1$;
- практически все геометрические фракталы задаются в форме бесконечной рекурсии, например, треугольник Серпинского:

В программировании рекурсия – это вызов функции из неё же самой, непосредственно (простая рекурсия) или через другие функции (сложная или косвенная рекурсия), например, функция A вызывает функцию B, а функция B – функцию A. Количество вложенных вызовов функции или процедуры называется *глубиной рекурсии*.

Определение функции и структура программы

Преимущество рекурсивного определения объекта заключается в том, что такое конечное определение теоретически способно описывать бесконечно большое число объектов. С помощью рекурсивной программы возможно описать бесконечное вычисление, причём без явных повторений частей программы.

Функции и структура программы

```
int fib(int i) {
  if (i < 3) return 1;
  return fib(i-1) + fib(i-2);
}
int main() {
  for(int i = 1; i < 11; i++)
    printf("F(%d) = %d\n", i, fib(i));
  return 0;
}
```

```
F(1) = 1
F(2) = 1
F(3) = 2
F(4) = 3
F(5) = 5
F(6) = 8
F(7) = 13
F(8) = 21
F(9) = 34
F(10) = 55
```

Пример. Программа, демонстрирующая взаимную рекурсию. Эта программа определяет, четно ли количество единиц в двоичном представлении заданного числа n (1 – четно, 0 – нечетно). Рекурсивная функция g0() обрабатывает состояние «четное количество» и завершается с результатом 1, а рекурсивная функция g1() обрабатывает состояние «нечетное количество» и завершается с результатом 0.

Функции и структура программы

```
int g0 (unsigned n) {
  if( n == 0) return 1;
  if( (n & 01) == 0) return g0(n >> 1);
  return g1(n >> 1);
}
int g1 (unsigned n) {
    if( n == 0) return 0;
    if( (n & 01) == 0) return g1(n >> 1);
    return g0(n >> 1);
}
int main() {
  unsigned n;
  while(1) {
    printf("enter the number (>0): "); scanf("%d", &n);
    if( n == 0) break;
    printf("result=%d\n", g0(n));
  }
  return 0;
}
```

```
enter the number (>0): 10
result=1
enter the number (>0): 7
result=0
enter the number (>0): 5
result=1
enter the number (>0): 4
result=0
enter the number (>0): 0
```

The end

Спасибо за внимание !