Лабораторная работа №7

Функции

Цель работы: Изучение теоретических сведений и получение практических навыков по работе с функциями.

Объявление и прототип функции

Функции предназначены для упрощения читаемости программы и представляют собой группу операторов, которая выполняет законченное действие. Функции позволяют использовать многократно один и тот же программный код.

В языке С функции являются единственным видом подпрограмм в отличие от языков Pascal, Fortran и т.п. Функции дают удобный способ заключения некоторой части вычислений в черный ящик, который в дальнейшем можно использовать, не интересуясь его внутренним содержанием. Использование функций является фактически единственным способом справиться с потенциальной сложностью больших программ и реализовать *принцип модульного программирования*. В основе принципа модульного программирования лежит разделение программы на автономные модули (фрагменты).

Для использования функции достаточно знать ее интерфейс: список передаваемых в функцию параметров и тип возвращаемого функцией значения. Интерфейс функции полностью определяется ее прототипом (объявление функции) и в общем виде может быть представлен в форме:

```
Тип_возвращаемого_значения Имя_функции (список параметров);
```

При использовании прототипа функции не обязательно указывать имена передаваемых параметров, достаточно списка типов переменных. Объявление прототипа всегда заканчивается точкой с запятой.

Объявление функции в общем случае будет иметь следующий вид:

```
Тип_возвращаемого_значения Имя_функции (список параметров)
{
      Операторы;
      return возвращаемое_значение;
}
```

Таким образом, прототип функции – это ее опережающее значение. Основное преимущество прототипа состоит в том, что опережающие объявления дают возможность программировать вызов функции, которая в данной точке программы еще не задана. При объявлении прототипа – он всегда предшествует функции `main()`, чтобы при компиляции

программы компилятор смог идентифицировать использованную в основной программе функцию, описание которой следует за функцией `main()`.

Фактические и формальные параметры

В языке С все аргументы функций передаются «по значению». Это означает, что вызванная функция получает значения своих аргументов с помощью временных переменных (фактически через стек), а не их адреса. Таким образом, в языке С вызванная функция не может изменить переменную из вызывающей функции: она может менять только свою собственную временную копию.

Имена, использованные для аргументов функции, являются чисто локальными и недоступны никаким другим функциям: другие функции могут использовать те же самые имена без возникновения конфликта.

Определение функции в списке аргументов содержит *формальные параметры*, значения которых неизвестны на этапе написания тела функции. Они приобретают конкретное значение в момент вызова функции, когда в скобках указываются *фактические параметры*, т.е. переменные или константы, имеющие конкретное значение.

Следует отметить, что в списке формальных параметрах необходимо указывать тип для каждой переменной, т.е. `float x, float y, int t`. Попытка использовать сокращение `float x,y` автоматически приведет к ошибке при компиляции.

Рассмотрим, например, функцию вычисления суммы двух переменных типа `float`. Ниже приведены два листинга программы вычисления суммы – с использованием прототипа и без его использования.

С прототипом функции

```
#include<stdio.h>
float sum(float, float);
int main()
{
  float a,b,s;
  printf(“\n Input a,b: \n”);
  scanf(“%f%f”,&a,&b);
  s=sum(a,b);
  printf(“\n %f + %f = %f\n”, a,b,s);
  return 0;
}
float sum(float a, float b)
{     return a+b;
}
```

Без прототипа функции

```
#include<stdio.h>
float sum(float a, float b)
{
      return a+b;
}
int main()
{
  float a,b,s;
  printf(“\n Input a,b: \n”);
  scanf(“%f%f”,&a,&b);
  s=sum(a,b);
  printf(“\n %f + %f = %f\n”,a,b,s);
  return 0;
}
```

Отметим, что во втором случае объявление прототипа не требуется поскольку, описание функции предшествует `main()` и компилятор находит его прежде, чем встречает вызов функции `sum` в основной программе `main()`.

Возвращаемое значение

Помимо объявления функции в программе должно в обязательном порядке присутствовать определение функции (тело функции). Тело функции представляет собой блок, заключенный в фигурные скобки, и содержащий выполняемые операторы и ключевое слово `return`. Ключевое слово `return` позволяет вернуть значение, вычисленное в теле функции, в основную программу. Тип возвращаемого значения должен соответствовать заявленному типу функции. Обратите внимание, что любой оператор, который будет следовать после `return`, не будет выполняться. В качестве аргумента оператора `return` можно написать любое выражение. Функция не обязана возвращать какое-либо значение ; оператор `return`, не содержащий никакого выражения, приводит к такой же передаче управления, как «сваливание на конец» функции при достижении конечной правой фигурной скобки, но при этом в вызывающую функцию не возвращается никакого полезного значения.

Вызов функции

Как показано в приведенном выше примере, вызов функции организован через оператор присваивания:

```
s=sum(a,b);
```

причем тип переменной `s` и тип возвращаемого значения функции совпадают (`float`).

В случае, если функция имеет тип `void` (пустой), то вызов функции может быть организован без присваивания. Рассмотрим на примере функции выводящей на экран таблицу:

```
#include<stdio.h>

void fill(int N)
{
      int i;  // локальная переменная
      for (i=1; i<=N; i++)
            printf(«\n %10d %15d»,i,i*i);
      printf(«\n»);  // функция не возвращает значение в main()
}
int main()
{
      int N;
      printf(“\n Input N:”);
      scanf(“%d”,&N);
      fill(N);  // мы вызываем функцию
      return 0;
}
```

Отметим также, что при выборе имен функций следует руководствоваться теми же правилами, как и при выборе имен переменных. Не следует использовать зарезервированные имена (совпадающие с функциями языка С) для пользовательских функций, чтобы не вызывать конфликтов имен.

Передача параметров по ссылке

При необходимости все же можно добиться, чтобы вызываемая функция изменила переменную из вызывающей функции. Для этого вызывающая функция должна обеспечить установление адреса переменной (технически, через указатель на переменную), а в вызываемой функции надо описать соответствующий аргумент как указатель и ссылаться к фактической переменной косвенно через него. Мы рассмотрим это подробнее в дальнейшем.

```
#include<stdio.h>
void func(float a, float b, float *res)
{
  *res=a+b;
}
int main()
{
  float a,b,s;
  printf("\n Input two float numbers: \n");
  scanf("%f%f",&a,&b);
  func(a,b,&s);
  printf("\n %.2f + %.2f = %.2f\n",a,b,s);
  return 0;

}
```

Фактически здесь мы передаем в функцию указатель, при этом при вызове нам приходится применять операции разадресации и взятия адреса явным образом (при вызове функции). Чтобы внутри функции изменить значение переменной, применяется операция получения значения по адресу. В данном случае указатель `*res` указывает на область памяти, в которой хранится переменная `s`.

Контрольные вопросы

1. Дайте определение функции. Что такое тип возвращаемого значения, список аргументов, тело функции. Приведите примеры.
2. Примеры функций, которые не возвращают значений, и функций без аргументов.
3. Как происходит вызов функции и возврат из нее. Оператор `return`. Примеры.
4. Формальные и фактические параметры функции. Соответствие типов. Примеры.
5. Что такое вызов функции «по значению» и «по ссылке»?
6. Может ли функция изменить значение своего аргумента в вызывающей функции? Обоснуйте ответ.

7. Для чего в программе используется прототип функции? В каких случаях он используется? Приведите примеры.
8. Найдите ошибки в следующих конструкциях

а)
```
int F(int k)
{
      float s=(float)k;
      return s;
}
```

б)
```
float  res(float k, float t)
{
      float s=k-t;
      return;
}
```

в)
```
void  division(int, int)
{
      printf("%d%d",int,int);
}
```

г)
```
int Sum (int a,b)
{
      int s=a+b;
      return s;
}
```

д)
```
void printf(char s)
{
      printf("%c",s);
}
```

е)
```
float main(int k);
{
      return k;
}
```

9. Рассмотрите следующие определения функций. Дайте их краткую характеристику, сформулируйте основную задачу, которую выполняют данные функции:

а)
```
int min (int x, int y)
{
      if (x<y)
            return x;
      else
            return y;
}
```

б)
```
float pow(float x, int n)
{     int i;
      float p=1.0;
      for(i=0;i<abs(n);i++)
            p*=x;
      return p;
}
```

в)
```
void Print_hello(void)
{
      printf("\nHello!\n");
}
```

г)
```
void print(float x)
{
      printf("\nx=%f\n",x);
}
```

д)
```
long fact(int n)
{
      int p=1,i;
      for(i=2;i<=n;i++)
            p*=i;
      return p;
}
```

е)
```
float seq(int n)
{     int i=n;
      float s=0.0,t;
      do{
            t=1.0/i;
            s+=t;
            i*=n;
      }while(t>1e-10);
      return s;
}
```

ж)
```
void repeat(char v, int n)
{
      int i;
      for(i=0;i<n;i++)
            printf("\n%c\n",v);
}
```

з)
```
float mult(float x, float y)
{
      return x*y;
}
```