

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННО БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Учебный Центр Информационных Технологий «Информатика»



Лабораторная работа № 4
по дисциплине «Информатика и программирование 2 часть»

Выполнил слушатель: Пешков Е.В.

Вариант: 1

Дата сдачи:

Преподаватель: Юшманов А.А.

Новосибирск, 2019г.

1. Цель

Научиться решать задачи с использованием строк.

2. Вариант задания

Разработать функцию с переменным числом параметров. Для извлечения параметров из списка использовать операцию преобразования типа указателя.

Вариант 1

Целая переменная - счетчик, затем последовательность вещественных переменных. Функция возвращает сумму переменных.

Теория

Преобразование указателей

Особенности работы с памятью в Си

Если под управлением памятью понимать размещение в ней данных, то в большинстве языков высокого уровня программист избавлен от этой участи. Более того, сама задача эффективного размещения данных в памяти относится к области системного программирования, поэтому в средах прикладного программирования эти механизмы присутствуют как данность и скрыты. Поскольку единственным «представителем» памяти в программе выступают переменные, то управление памятью определяется тем, каким образом работает с ними и с образованными ими структурами данных язык программирования. Большинство языков программирования компилирующего типа однозначно закрепляет за переменными их типы данных и ограничивает работу с памятью только теми областями, в которых эти переменные размещены. Программист не может выйти за пределы самим же определенного шаблона структуры данных. С другой стороны, это позволяет транслятору обнаруживать допущенные ошибки, как в процессе трансляции, так и в процессе выполнения программы.

Наличие указателей или ссылочных типов (Паскаль, Си, Java) а также динамических переменных позволяет «вручную» реализовать динамические структуры данных средствами языка программирования, однако принципиально не меняет систему взаимоотношений и меру ответственности программиста и транслятора.

В языке Си ситуация принципиально иная по двум причинам:

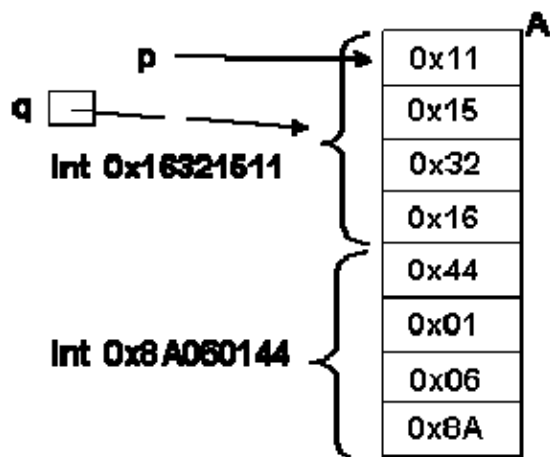
- наличие операции адресной арифметики при работе с указателями позволяет, в принципе, выйти за пределы памяти, выделенной транслятором под указуемую переменную и адресовать память как «до» так и «после» нее. Другое дело, что это должно производиться осознанно и корректно;
- присваивание и преобразование указателей различных типов, речь о котором пойдет ниже, позволяет рассматривать одну и ту же память «под различным углом зрения» в смысле типов заполняющих ее переменных.

Преобразование типа указателя

Операцию присваивания указателей различных типов следует понимать как копирование адреса памяти, содержащегося в указателе. Иначе говоря, назначение указателя в левой части на ту же самую область памяти. Но поскольку тип указуемых переменных у них разный, то эта область памяти по правилам интерпретации указателя будет рассматриваться как заполненная переменными либо одного, либо другого типа.

```
char A[20]={0x11,0x15,0x32,0x16,0x44,0x1,0x6,0x8A};
char *p; int *q; long *l;
p = A; q = (int*)p; l = (long*)p;
p[2] = 5;           // записать 5 во второй байт области A
q[1] = 7;           // записать 7 в первое слово области A
```

Здесь **p** - указатель на область байтов, **q** - на область целых, **l** - на область длинных целых. Соответственно операции адресной арифметики ***(p+i)**, ***(q+i)**, ***(l+i)** или **p[i]**, **q[i]**, **l[i]** адресуют **i**-ый байт, **i**-ое целое и **i**-ое длинное целое от начала области. область памяти имеет различную структуру (байтовую, словную и т.д.) в зависимости от того, через какой указатель мы с ней работаем. При этом неважно, что сама область определена как массив типа **char** - это имеет отношение только к операциям с использованием идентификатора массива.



Указатели различных типов в общей памяти

Присваивание значения указателя одного типа указателю другого типа сопровождается действием, которое называется в Си **преобразованием типа указателя**, и которое в Си++ обозначается всегда явно. Операция **(int*)p** меняет в текущем контексте тип указателя **char*** на **int***. На самом деле это действие является чистой фикцией (команды транслятором не генерируются). Транслятор просто запоминает, что тип указываемой переменной изменился, поэтому операции адресной арифметики и косвенного обращения нужно выполнять с учетом нового типа указателя. Преобразование типа указателя можно выполнить не только при присваивании, но и внутри выражения, «на лету». Операция **явного преобразования типа указателя** меняет тип указываемого элемента только в цепочке выполняемых операций.

```
char A[20]; ((int *)A)[2] = 5;
```

Имя массива **A** - указатель на его начало - имеет тип **char***, который явно преобразуется в **int***. Тем самым в текущем контексте мы ссылаемся на массив как на область целых переменных. Применительно к указателю на массив целых выполняется операция индексации и последующее присваивание целого значения 5 во второй элемент целого массива, размещенного в **A**. Операция ***p++** применительно к любому указателю интерпретируется как «взять указываемую переменную и перейти к следующей», значением указателя после выполнения операции будет адрес переменной, следующей за выбранной. Использование такой операции в сочетании с явным преобразованием типа указателя позволяет извлекать или записывать переменные различных типов, последовательно расположенных в памяти.

```
char A[20], *p=A;
*p++ = 5;           // Записать в массив байт с кодом 5
*((int*)p)++ = 5;   // Записать в массив целое 5 вслед за ним
*((double*)p)++ = 5.5; // Записать в массив вещественное 5.5 вслед за ним
```

Замечание: операция ++ в сочетании с преобразованием типа указателя «на лету» принимается не всеми трансляторами. В случае неудачи необходимо использовать промежуточное присваивание со сменой типа указателя.

3. Анализ задачи и алгоритм

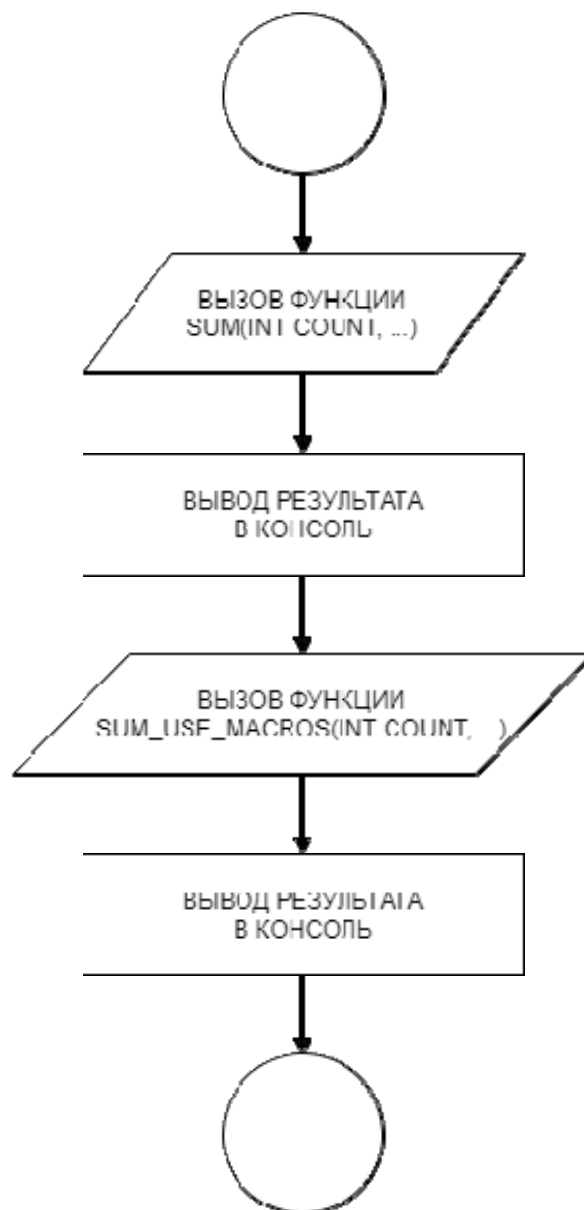
1) Анализ задачи

Входные данные: количество аргументов, последовательность вещественных аргументов.

Результат: сумма вещественных аргументов.

Метод решения: устанавливаем указатель на первый аргумент и определяем количество вещественных аргументов, выполняем преобразование указателя к вещественному типу.

2) Алгоритм решения задачи



4. Описание программной реализации

1) Используемые функции

Функция **double sum (int count, ...);**

Аргументы функции:

int count – количество аргументов передаваемых в функцию.

Возвращаемый результат: сумма переданных.

Принцип работы: вычисляет сумму переданных аргументов.

Функция **double sum_use_macros (int count, ...);**

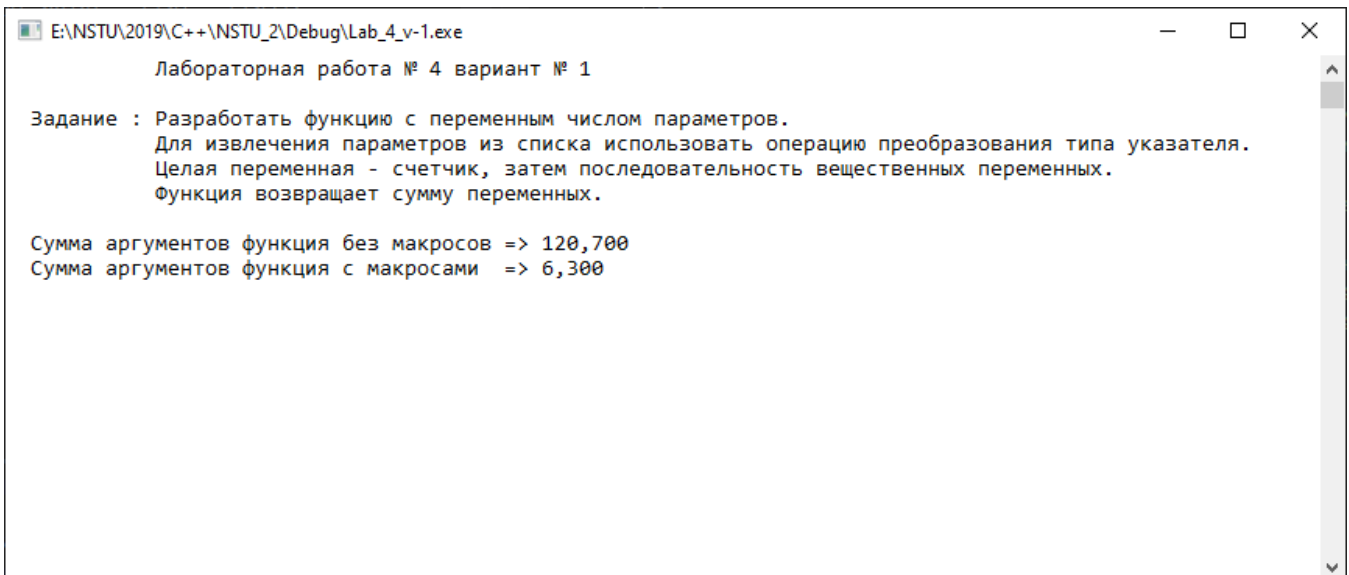
Аргументы функции:

int count – количество аргументов передаваемых в функцию.

Возвращаемый результат: сумма переданных.

Принцип работы: вычисляет сумму переданных аргументов.

5. Пример работы программы



```
E:\NSTU\2019\C++\NSTU_2\Debug\Lab_4_v-1.exe
Лабораторная работа № 4 вариант № 1
Задание : Разработать функцию с переменным числом параметров.
Для извлечения параметров из списка использовать операцию преобразования типа указателя.
Целая переменная - счетчик, затем последовательность вещественных переменных.
Функция возвращает сумму переменных.
Сумма аргументов функция без макросов => 120,700
Сумма аргументов функция с макросами => 6,300
```

6. Выводы

В ходе выполнения лабораторной работы были изучены строки и указатели и применены в решении задачи.

Приложение. Текст программы

```
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <locale.h>

double sum(int count, ...);
double sum_use_macros(int count, ...);

int main()
{
    setlocale(LC_ALL, "RU-ru");

    printf("                Лабораторная работа № 4 вариант № 1\n\n");
    printf("    Задание : Разработать функцию с переменным числом параметров.\n");
    printf("    Для извлечения параметров из списка использовать операцию преобразования\n");
    printf("    типа указателя.\n");
    printf("    Целая переменная - счетчик, затем последовательность вещественных пере-\n");
    printf("    менных.\n");
    printf("    Функция возвращает сумму переменных.\n\n");
    printf("    Сумма аргументов функция без макросов => %.3f\n", sum(3, 50.1, 40.3, 30.3));
    printf("    Сумма аргументов функция с макросами  => %.3f\n", sum_use_macros(4, 2.1, 1.1, 3.1,\nNULL));
    getchar();
    return 0;
}

double sum(int count, ...)
{
    double result = 0;
    for (int* ptrArg = &count; count > 0; count--)
    {
        ptrArg++;
        double ptrDouble;
        ptrDouble = *(double*)ptrArg++;
        result += ptrDouble;
    }
    return result;
}

double sum_use_macros(int count, ...)
{
    double result = 0;
    va_list sumArgs;
    va_start(sumArgs, count);
    for (int i = 0; i < count; i++)
    {
        result += va_arg(sumArgs, double);
    }
    va_end(sumArgs);
    return result;
}
```