Семинар 4. Определения, описания и вызовы функций; рекурсивные функции; inline-функции; перегрузка функций; функции с переменным количеством параметров

1. Определения, описания и вызовы функций

Функция в Си++ – это основное понятие, без которого невозможно обойтись. Каждая программа должна включать функцию main, являющуюся точкой входа в откомпилированную программу.

Точка входа – то самое место, откуда начинается исполнение кода программы.

1. Определения, описания и вызовы функций

Каждая программа в Си++ – совокупность функций, каждая из которых должна быть определена или по крайней мере описана до её использования.

1. Определения, описания и вызовы функций

Определение функций	Описание функций (прототип)
1. Тип возвращаемого функцией значения (тип результата); 2. Имя функции; 3. Совокупность параметров (включая их тип; параметры перечисляются через запятую); 4. Последовательность действий, выполняемых при её вызове (тело);	1. Тип возвращаемого функцией значения (тип результата); 2. Имя функции; 3. Совокупность параметров (тип указывать обязательно, имена параметров – необязательно; параметры перечисляются через запятую);
тип имя_функ(специф_парам) тело_функции	тип имя_функ(специф_парам);

Как в определение, так и в описание функций может входить спецификация исключений, генерация которых запланирована в теле функции.

1. Определения, описания и вызовы функций

Примеры определения функций:

1. Функция ничего не возвращает (типа void), печатает значения name и value

```
void print (char * name, int value)
{cout<<'\n'<<name<<'\t'<<value;}
```

Здесь отсутствует оператор return, хотя он может присутствовать в форме return;

1. Определения, описания и вызовы функций

Пример вызова этой функции:

```
char * str = "Vanya";
…
print(str, 31);
…
int hb2u = 31 + 1;
print(str, hb2u)
```

1. Определения, описания и вызовы функций

Примеры определения функций:

2. Функция нахождения максимального из двух чисел

```
double max (double a, double b)
{return a > b ? a : b;}
```

или

```
double max (double a, double b){
    if (a > b) return a;
    else return b; // else можно опустить
}
```

1. Определения, описания и вызовы функций

Пример вызова этой функции:

```
//максимум из двух чисел
double maxValue = max (i, j);

double maxValue3 = max(i, max(j, k)); // из 3

// максимум из четырёх чисел
double maxValue4 = max(max(i, j), max(k, l));
```

1. Определения, описания и вызовы функций

Примеры определения функций:

3. Функция без параметров

```
void starRow20 (void){
	for(int i = 0; i < 20; i++){
		cout << '*';
	}
}
```

Список параметров мог быть пустым. Отсутствие параметров эквивалентно void.

1. Определения, описания и вызовы функций

Пример вызова этой функции:

```
starRow20();
```

1. Определения, описания и вызовы функций

Примеры определения функций:

4. Функция, в которой спецификация параметра содержит значение по умолчанию

```
void printStr(char * str1="Hello, ", char * str2="world")
{
    cout << str1 << str2;
}
```

1. Определения, описания и вызовы функций

Пример вызова этой функции:

```
printStr(); // Выведет Hello, world
printStr("Hi, "); // Выведет Hi, world
printStr("123", "456"); // Выведет 123456

printStr(, "Moscow") // Недопустимо
// т.к. отсутствует первый аргумент
```

1. Определения, описания и вызовы функций

Примеры определения функций:

5. Функция, вычисляющая НОД (наибольший общий делитель, greatest common divisor)

Алгоритм Евклида:

- если x == y, то ответ найден, НОД = x
- если x < y, то y заменяется значением y – x
- если x > y, то x зам. знач. x - y

1. Определения, описания и вызовы функций

Примеры определения функций:

5. Функция, вычисляющая НОД (наибольший общий делитель, greatest common divisor)

```
int GCD(int x, int y)
{
    while(x!=y){
        if(x > y) x = x – y;
        else y = y – x;
    }
    return x;
}
```

1. Определения, описания и вызовы функций

При обращении к функции параметры заменяются аргументами, причём соблюдается строгое соответствие по типам. Проверка соответствия типов аргументов и параметров выполняется на этапе компиляции.

1. Определения, описания и вызовы функций

Примеры описания функций:

```
void print(char *, int);

double min(double a, double b);
или
double min(double, double b);
// опустили имя одного из параметров
```

2. Рекурсивные функции

Рекурсивная функция – это такая функция, которая вызывает сама себя (в простейшем случае). При создании рекурсивной функции всегда должно быть указано условие прекращения такого вызова – т.н. база рекурсии.

2. Рекурсивные функции

Пример 1. Вычисление факториала числа.

```
long fact(int k){
    if (k<0) return 0;
    if (k == 0) return 1;
    return k*fact(k-1);
}
```

2. Рекурсивные функции

Пример 2. Вычисление n-го члена ряда Фиббоначи.

```
long fib(int k){
    if ((k == 1) || (k == 2)) return 1;
    else return fib(k-1)*fib(k-2);
}
```

2. Рекурсивные функции

Задания.

1. Создать нерекурс. функцию получения максимального из четырёх чисел.
2. Создать рекурсивную функцию получения суммы всех чисел от 1 до n, n – параметр функции.
3. Создать рекурсивную функцию получения n-го члена ряда Триббоначчи, в котором F(n)=F(n-1) + F(n-2) + F(n-3),
   F(1) = 1, F(2) = 1.
4. Создать рекурсивную и нерекурсивную функции, которая возвращает true, если число простое, и false иначе.
5. Создать функцию поиска наименьшего общего кратного двух чисел. Наименьшее общее кратное – это наименьшее натуральное число, которое делится на оба числа без остатка.

3. inline-функции

Некоторые функции в языке Си++ можно определить с помощью специального служебного слова inline. Компилятор в каждое место вызова функции помещает соответствующим образом настроенные команды кода операторов тела функции.

При многократных вызовах размеры программы могут увеличиться, но исключаются затраты на передачи управления.

3. inline-функции

Причины, которые могут препятствовать реализации функции как подставляемой, даже если она определена со спецификатором inline:

1. Функция слишком велика, чтобы выполнить подстановку
2. Функция рекурсивна
3. Обращение к функции в программе размещено до определения
4. Функция вызывается более одного раза в выражении

3. inline-функции

Пример. Вычисление расстояния от начала координат (точки (0, 0)) до точки с координатами (x, y).

```
inline double rasst(double x=0, double y=0){
    return sqrt(x*x + y*y);
}
```

4. Перегрузка функций

Перегруженные функции имеют одинаковое имя, но различаются по типам и/или количеству параметров.

4. Перегрузка функций

Пример.

```
#include <iostream>
using namespace std;
int sum(int a, int b) {return a+b;}
double sum(double a, double b){return a+b;}
int sum(int a, int b, int c){return a+b+c;}
int main() {
        cout << sum(1, 2) << endl;
        cout << sum (1, 2, 3) << endl;
        cout << sum(1.5, 2.0) << endl;
return 0;
}
```

5. Функции с переменным количеством параметров

Допустимы функции, количество параметров у которых при компиляции не фиксировано. Могут быть неизвестны и типы параметров.

Количество и типы параметров становятся известными только в момент вызова функции, когда явно задан список аргументов.

5. Функции с переменным количеством параметров

При определении и описании таких функций, имеющих списки параметров неопределённой длины, спецификация параметров заканчивается многоточием.

Формат:
тип имя(спец_явных_параметров, …);

5. Функции с переменным количеством параметров

Для работы с параметрами:

```
#include <cstdarg>
```

В теле функции для обращения к аргументам обязательно определить объект типа va_list:

```
va_list p;
```

5. Функции с переменным количеством параметров

Объект типа va_list необходимо связать с первым необязательным параметром. Это достигается следующим образом:

va_start(p, последний_явный_параметр);

Именно для этого функция с переменным количеством параметров должна иметь хотя бы один явно специфицированный параметр.

5. Функции с переменным количеством параметров

С помощью разыменования указателя типа va_list мы можем получить значение первого аргумента из переменного списка параметров. Однако нам должен быть известен тип этого аргумента. Тип параметра должен быть передан в функцию каким-либо образом. Если известен тип аргумента, то значение аргумента доступно с помощью `*(type *) p` или `va_arg(p, type)`

5. Функции с переменным количеством параметров

Пример. Функия суммирования с переменным количеством параметров.

```
#include <iostream>
#include <cstdarg>
using namespace std;
long summa(int k, ...){
        va_list p;
        va_start(p, k);
        long sum = 0;
        for(;k;k--)
                sum += va_arg(p, int);
        return sum;
}
int main() {
        cout << summa(6, 2, 3, 4, -10, -20, -30) << endl;
        cout << summa(2, 4, 3) << endl;
        return 0;
}
```

5. Функции с переменным количеством параметров

Пример. Функия суммирования с переменным количеством параметров.

```
#include <iostream>
#include <cstdarg>
using namespace std;
long summa(int k, ...){
        va_list p;
        va_start(p, k);
        long sum = 0;
        for(;k;k--){
                sum += *((int *) p);
                p = p + 4; // смещение на 4 обеспечивает смещ. на 4 байта
        }
        return sum;
}
int main() {
        cout << summa(6, 2, 3, 4, -10, -20, -30) << endl;
        cout << summa(2, 4, 3) << endl;
        return 0;
}
```

5. Функции с переменным количеством параметров

Для возврата к началу переменного списка параметров необходимо применение va_end(). Единственным параметром va_end() является указатель типа va_list. va_end() обычно модифицирует свой аргумент, поэтому указатель типа va_list нельзя будет использовать повторно без предварительного вызова va_start().

5. Функции с переменным количеством параметров

Задания.

1. Определить функцию с переменным количеством аргументов. В качестве первого параметра передать количество последующих аргументов. Функция должна возвращать среднее от значений самого наименьшего и наибольшего аргументов (кроме первого) за вычетом среднего арифметического всех аргументов (кроме первого).