ЯП

# Правила функций

1. Функция решает тольку одну задачу
2. Все исходные данные попадают только через параметры
3. Все результаты возвращаются в вызывающую функцию

- 1 результат через RETURN

- доп результат через ссылки

- bool return только при условиях, а не успешности функции

- void при пустом ответе (полностью пустой, без ссылок)



# Перегрузка функций. Шаблоны функции.

Называться функции могут одинаково, но иметь разные данные

Использование разных функций с одинаковым именем с разными параметрами называется **перегрузкой. (Тип функции в перегрузке не участвует)**

double pow (double, int)

double pow (double, double)

float pow (float, float)

float pow (float, int)

**Примеры перегрузок:**

void Out (int\* mas, int n);

void Out (double\* mas, int n);

void Out (long\* mas, int n);

void Out (float \* mas, int n);

void Out (float\* mas, int n, int m);

Вызов должен быть однозначным – компилятор определяет, какая из функци вызвана по набору аргументов.

**Неоднозначность может быть вызвана:**

1. При преобразовании типа
2. При использовании параметров ссылок

void Func (int A)

void Func (int &R)

int main(void)

{

int a=7

Func(a);

...

}

1. При использовании параметров по-умолчанию

Неоднозначность можно преодалеть, упоминая тип данных

Многие алгоритмы не зависят от типов данных, с которыми они работают.

С помощью шаблона функции можно определить алогритм, который будет применяться к данным различного типа.

Конкретный тип данных передается функции в виде параметра (явно или по умолчанию) на этапе компиляции.

# Шаблоны функции

template <class **Type**>

Имя функции

{

Тело функции

}

С типом Type могут объявляться параметры внутренние локальные переменные.

Комплятор, глядя на первый вызов, сравнивая параметр с аргументом, определяет что тип Type=int. Формируется **экземпляр функции** путем замены формального типа (Type) на int. Те же самые процессы происходят и для последующих вызовов.

Шаблоны функции должны быть физически в том же файле, что и функция main().

Для этого тексты шаблонов помещаем в заголовочный файл (например, Max.h) и до функции main() пишем:

#include <Max.h>

int main(void)

{

...

}

Объявления шаблонов функции не используем.

# Потоки

ios – базовый потоковый класс

isstream – класс входных потоков

ostream – клас выходных потоков

iostream – класс двунаправленных потоков

**ifstream** fin – класс входных файловых потоков

**ofstream** fout – класс выходных файловых потоков

**fstream** f – класс файловых потоков ввода/вывода

**Необходимо подключать:**

#include <iostream>

using namespace std;

#include <fstream>

#include <iomanip>

Для работы с файлами необходимо выполнить следующие действия:

1. Создать поток
2. Связать поток методом open()
3. Произвести обмен с потоком
4. Закрыть поток

...open(char\*file,int mode, int access);

file – путь к файлу

int mode – режим открытия

# Режимы открытия

**ios::app** – вывод происходит всегда в конец файла

**ios::ate** – указатель потока при открытии позиционируется в конец файла, в дальнейшем позиционирование можно изменить

**ios::trunc** – при открытии файла содержимое уничтожается (вывод)

**ios::\_Nocreate** – открыть существуеющий файл, если не существует – ошибка

**ios::\_Noreplace** – создать новый файл, если существует- ошибка

**ios::binary** – открыть в двоичном коде

**ios::in** – поток для вывода

**ios::out** – поток для вывода

Синтаксис

fin.open(“f1.txt”,ios::\_Nocreate);

if(!fin){

cout<<”Ошибка!”;

}

# Обмен потоками

**Посимвольно**

С игнорированием пробельных символов:

Ввод:

char simv;

fin.get(simv);

Вывод:

fout.put(simv);

Посимвольно, игнорируя пробельные символы:

**Построчно**

Построчно, включая пробелы:

Ввод:

char str[80]

fin.getline(str,80);

Вывод:

fout<<str<<endl;

**Словами, т.е. строки без пробелов:**

Ввод:

char slovo [30];

fin>>slovo;

Ввод:

fout<<slovo<<’’;

**Числами**

double a;

fin>>a;

fout<<a;

**Двоичный ввод-вывод**

char str[80];

fin.read(str,strlen(str));

fout.write(str,strlen(str));

fin.seekg(-1,ios::cur);

-Смещает указатель потока на один символ назад

long l;

l=fin.tellg();

-Определяет позицию указателя

# Цикл обработки исходного файла

while (--Чтение){

//обработка порции

}

**Методы состояния потоков**

**fin.eof()** – ненулевое значение-конец файла

**fin.good()** – нет ошибок (ненулевое значение)

**fin.fail()**

**fin.bad()**

**Закрытие потока**

fin.close();

fout.close();

# Обработка сиволов строк

strlen() – реальный размер строки

strcpy() – копирует одну строку в другую

strncpy() – копирует n символов одной строки в другую

strcat() – дозапись одной строки в конец другой

strncat() – дозапись n символов одной строки в другую

strstr() – поиск строки в строке

strcmp() – сравнение на равенство двух строк

while(f.get(s)){  
 if(s>=’0’&&s<=’9’){

f.seekg(-1,ios::cur);

f.put(‘\*’);

f.seekg(f.tellg());

}

}

f.close();

cout<<”\ngood!”;

return 0;

При построении функции в качестве параметра можно передавать имя файла или ссылку на поток, в первом случае нет уверенности, что нашелся и удачно читается, поэтому тип функции bool, а во втором (поток передаётся ссылкой) поток уже объявлен и инциализирован в вызывающей функции.

# Структуры

- запись состоящая из ряда полей расположенных в памяти подряд. Поля это даннные любого типа в т.ч. другие структуры.

Для использования необходимо:

1. Объявить шаблон структуры
2. Объявить переменные

struct book{

char name[20];

char author[20];

int year;

};

# Шаблон структуры

Схема, описывающая содержание структуры.

# Объединение

Используя объединение можно размещать данные из одного адреса в оперативной памяти. При этом в фиксированный момент времени может быть размещено только одно поле.

Union A{

Int a1;

Double a2;

Char a3;

}a;

A a;s