Функции

Директивы препроцессора

- Не совсем язык C
- Используются для “предварительной подготовки” программы
- Современные тенденции - отказ от них в C++

Директивы препроцессора

#define – определение макроса или препроцессорного идентификатора;

#include – включение текста из файла;

#undef – отмена определения макроса или идентификатора;

#line – смена номера следующей ниже строки;

#error – формирование текста сообщения об ошибке трансляции;

#pragma – действия, предусмотренные реализацией;

Директивы препроцессора

#if – проверка условия-выражения;

#ifdef – проверка определенности идентификатора;

#ifndef – проверка неопределенности идентификатора;

#else – начало альтернативной ветви для #if;

#endif – окончание условной директивы #if;

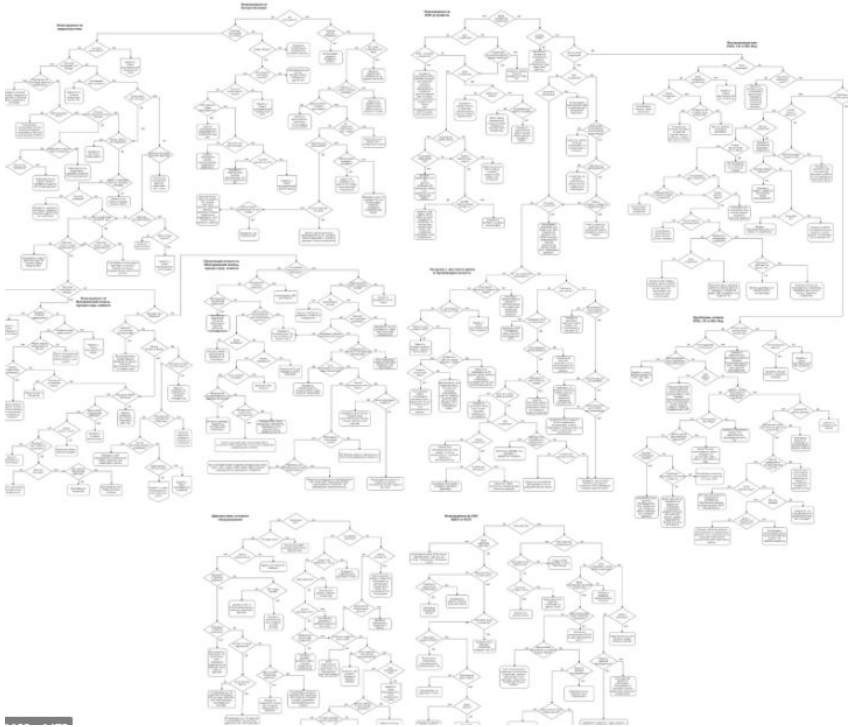
#elif – составная директива #else#if;

Проблемы сложных алгоритмов

- Иногда алгоритмы бывают непростыми…
- Циклы немного помогают, но не всегда…
- Простой вариант - надо ввести массив, поработать с ним и вывести ответ
- А если массивов много и разного размера?
- И работать надо не над одним, а над комбинациями?
- А как это всё тестировать?

Проблемы сложных алгоритмов

- А какая будет блок схема?



Решение - функции!

- Позволяют сильно упростить структуру программы
- Минимальный исполняемый модуль программы
- Обособленный, логически завершённый модуль
- Получает значение в виде параметров и, чаще всего, возвращает результат
- В программе всегда есть как минимум одна функция - main

Синтаксис

тип_возвращаемого_значения имя_функции (список_формальных_параметров)

{
тело_функции;
}

Пример

```
int pow2(int x)
{
	return x*x;
}

int main()
{
printf(“%d”,pow2(25));
return 0;
}
```

Прототип функции

- Описание функции без определения
- Используется в начале программы для указания параметров используемых функций
- Позволяет определить функцию позже, возможно даже в другом файле
- Пример:
  int pow2(int);

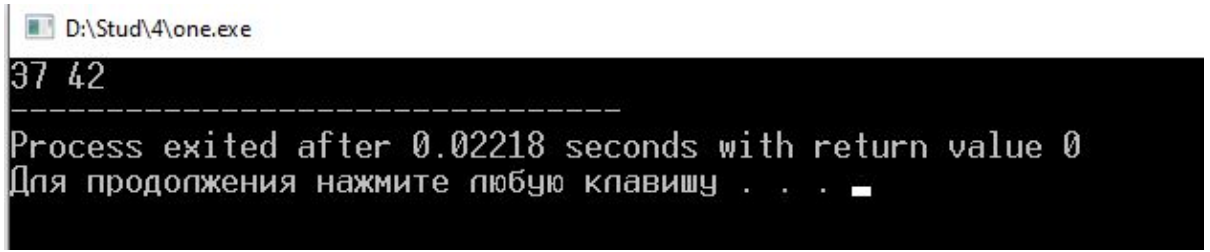
Формальные и фактические параметры

- Формальные указаны в определении функции
- Фактические - передаются при вызове
- Важно соблюдать типы, количество и порядок

Передача параметров в функцию

- В С передаётся только значение параметра, а не сама переменная!

```
int f(int a){
        a=a+5;
        return a;
}
int main(){
        int a=37;
        int b=f(a);
        printf(“%d %d”,a,b);
        return 0;
}
```



Область видимости переменных

- Всё сложно
- Обычные переменные видны внутри функции
- Глобальные - везде
- Внешние - между файлами

```
extern x;
int y;
int f1(){
        int z;
}
```

Как передать массив?

- У массива много значений, а передать можно только одно
- Размер массива не позволяет передать каждый элемент своим параметром
- Выход - указатели!
- Вместо значения передаётся адрес первого элемента и число элементов

Пример

```
void output(int *m, int n){
	for(int i=0;i<n;i++)
		printf(“%d”,m[i]); //либо так printf(“%d”,*(m+i));
	}

int main(){
	int mas[3]={1,2,3};
	output(mas,3);
	return 0;
}
```

Как передать двумерный массив

- Сложно!
- Важно соблюдать типы
- Важно помнить про отличие статического и динамического

Передача статического двумерного массива

```
void output(int *arr,int m, int n){
	for(int i=0;i<m;i++)
		for(int j=0;j<n;j++)
			printf(“%d ”,*(arr+i*n+j));
	}

int main(){
	int mas[3][3]={{1,2,3},{4,5,6},{7,8,9}};
	output(&mas[0][0],3,3);
	return 0;
}
```

Передача двумерного массива

```
void output(int arr[][3],int m, int n){
	for(int i=0;i<m;i++)
		for(int j=0;j<n;j++)
			printf(“%d ”,*(*(arr+i)+j));
	}

int main(){
	int mas[3][3]={{1,2,3},{4,5,6},{7,8,9}};
	output(mas,3,3);
	return 0;
}
```

Конец!