Лекция 2.
Функции и структуры.

План лекции

- Вспоминаем функции
- Параметры функций
- Структуры
- Переопределение операторов сравнения
- Переопределение арифметических операторов
- Переопределение ввода и вывода

Функции

Функция – это группа выражений, которой присвоено имя и которая может быть вызвана из какого-либо места в программе.

Функция может принимать параметры (данные извне) и возвращать какое-нибудь значение.

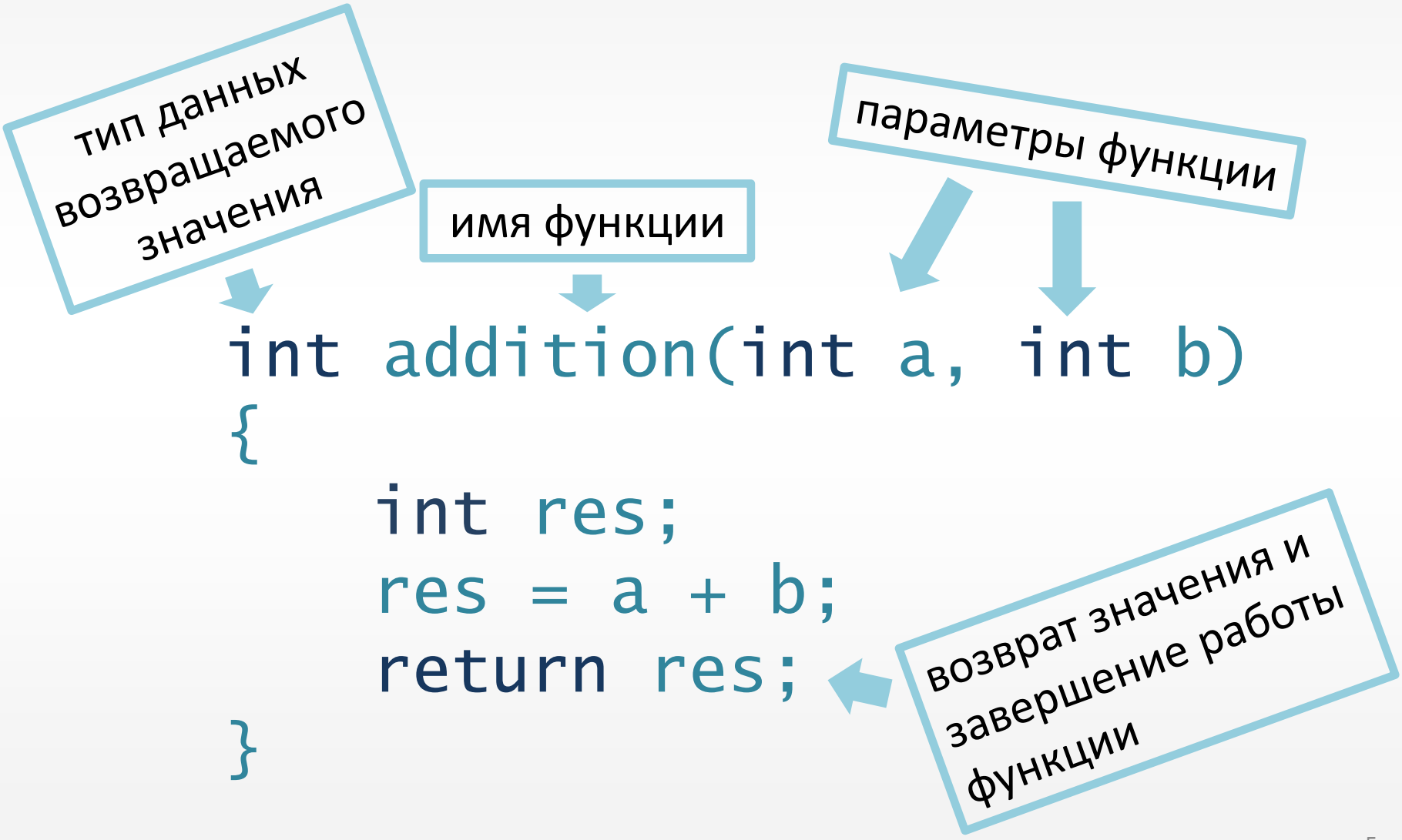
```
int addition(int a, int b)
{
    return a+b;
}
```

Функции

Наглядно можно представить функцию как станок на конвейерном производстве. Мы вызываем функцию – устанавливаем станок, передаём параметры – задаём исходные материалы и настройки, а после отработки, возможно, получаем на выходе результат.

```
int addition(int a, int b)
{
      return a+b;
}
```

Функции


тип данных возвращаемого значения
имя функции
параметры функции
int addition(int a, int b)
{
int res;
res = a + b;
return res;
}
возврат значения и завершение работы функции

Функции

Функция обязательно должна быть объявлена до (то есть выше) того, как её вызовут в коде программы, по той же причине, по которой надо объявлять переменные до работы с ними.

Функции

```
int addition(int a, int b)
{
     return a+b;
}

int main()
{
     …
     c = addition(1234, 5);
     …
}
```

ВЕРНО – функция объявлена до вызова

Функции

```
int addition(int, int);

int main()
{
     …
     c = addition(1234, 5);
     …
}

int addition(int a, int b)
{
     return a+b;
}
```

ТОЖЕ ВЕРНО – прототип функции объявлен до вызова

Функции

```
int main()
{
      …
      c = addition(1234, 5);
      …
}

int addition(int a, int b)
{
      return a+b;
}
```

НЕВЕРНО – функция объявлена после вызова

Функции

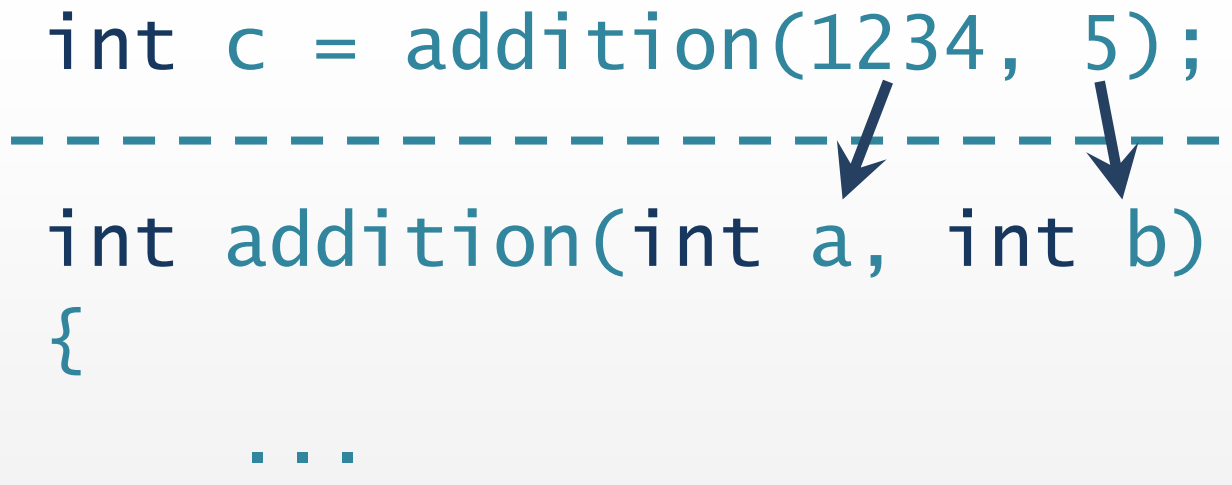
Функция может не возвращать никаких значений. Тогда она имеет тип данных `void`.

```
void write(int a)
{
    cout << “a = ” << a;
}
```

Если такая функция должна экстренно завершиться, пишут просто «`return;`», не передавая возвращаемого значения.

Параметры функции

Параметры функции – это некоторые значения, которые передаются извне при вызове функции. Для них автоматически создаются переменные (или константы).



Параметры функции

В качестве параметра функции можно передать массив.

```
int minimum(int mas[], int n)
{
	int res = mas[0];
	for(int i=1;i<n;i++)
	{
		if(res > mas[i])
			res = mas[i];
	}
	return res;
}
int main()
{
	int array[1000];
	…
	cout << minimum(array, 1000);
	…
}
```

Параметры функции

В качестве параметра можно передать даже другую функцию! Для этого используется такой тип данных, как указатель на функцию.

```
int foo((*pointer_name)(int, int))
{
      …
}
int func(int a, int b)
{
      …
}
```


```
int (*func_pointer)(int, int) = func;
int num = foo(func_pointer);
```

Структуры

Структура – это данные, сгруппированные под одним именем.

```
struct rabbit
{
    int weight;
    string colour;
    string name;
};
```

Структуры

Структуры как бы создают свой собственный тип данных, который может хранить объекты с множеством свойств, одинаковых для каждого объекта структуры.

```
struct rabbit
{
     int weight;          //Вес кролика
     string colour;       //Цвет кролика
     string name;         //Кличка кролика
} rbt;
```

Работа со структурами

```
struct rabbit
{
      double weight, puffyness;
      string name, colour;
} a, b, c;
rabbit d, e[50];
vector <rabbit> f;

d.weight = 3.4;
d.puffyness = 6;
d.name = “Elistrat”;
d.colour = “white”;
```

Объекты структуры можно объявлять сразу после объявления структуры, а также используя имя структуры как тип данных.

Самый простой способ задать значение параметрам структуры – явный: после создания объекта можно обратиться к каждому из его параметров и присвоить им нужные значения.

Работа со структурами

В структуру можно добавить так называемый конструктор – это функция внутри структуры, доступная только ей, которая автоматически присваивает заданные значения нужным параметрам структуры.

```
struct rabbit {
	double weight, puffyness;
	string name, colour;

	rabbit(int w, int p, string n, string c)
	{
		weight = w;
		puffyness = p;
		name = n;
		colour = c;
	}
};
```

Работа со структурами

```
struct rabbit {
      double weight, puffyness;
      string name, colour;

      rabbit(int w, int p, string n, string c)
      {
            weight = w;
            puffyness = p;        //В структуру можно
            name = n;       //добавить так называемый
            colour = c; //конструктор – это функция
      }                  //внутри структуры, доступная
};       //только ей, и автоматически присваивающая
  //заданные значения нужным параметрам структуры.
```

Работа со структурами

В таком случае в коде всё выглядит гораздо проще.

```
// С конструктором:
rabbit d(3.4, 6, “Elistrat”, “white”);

// Без конструктора:
rabbit d;

d.weight = 3.4;
d.puffyness = 6;
d.name = “Elistrat”;
d.colour = “white”;
```

Переопределение операторов сравнения

Оператор сравнения – это привычный нам оператор, который сравнивает два каких-либо значения и выдаёт в качестве ответа истину или ложь. Например, > - это оператор сравнения, и «5 > 8» - ложь.

Но структуры нельзя сравнивать, как числа, поэтому приходится писать своё «дополнение» к оператору сравнения – переопределять его для новой структуры.

> >= < <= ==

Переопределение операторов сравнения

Например, пусть мы будем мерять кроликов их пушистостью и весом:

```
struct rabbit{
    double weight, puffyness;
    string name, colour;

    bool operator<(const rabbit& a)
    {
        return weight * puffyness < a.weight * a.puffyness;
    }
};
```

```
if(a<b)
{
      …
}
```

Переопределение операторов сравнения

То же самое можно делать и с другими операторами:

```
struct rabbit{
    double weight, puffyness;
    string name, colour;

    bool operator<(const rabbit& a)
    {
        return weight * puffyness < a.weight * a.puffyness;
    }
    bool operator==(const rabbit& a)
    {
        return weight * puffyness == a.weight * a.puffyness;
    }
    bool operator<=(const rabbit& a)
    {
        return weight * puffyness <= a.weight * a.puffyness;
    }
};
```

Переопределение арифметических операторов

```
struct rabbit{
    double weight, puffyness;
    string name, colour;

    rabbit(int w, int p, string n, string c)
    {
        weight = w;
        puffyness = p;
        name = n;
        colour = c;
    }

    rabbit& operator=(const rabbit& a)
    {
        return rabbit(a.weight, a.puffyness, a.name, a.colour);
    }
};
```

В отличие от сравнений, арифметический оператор после выполнения действий должен возвращать не истину или ложь, а объект того же типа данных.

= + - * / % и т.д.

Переопределение операторов ввода и вывода

```
struct rabbit {
    …
    friend istream& operator>> (istream  &input, rabbit& a) {
        input >> a.name >> a.weight >> a.puffyness >> a.colour;
        return input;
    }
    friend ostream& operator<< (ostream &output, const rabbit& a) {
        output << a.name << " : " << a.weight << "kg, " << a.puffyness << "
puffy, " << a.colour;
        return output;
    }
}
int main() {
    rabbit a;
    cin >> a;
    cout << a;
    …
}
```

Переопределение операторов ввода и вывода

```
struct rabbit {
    …
}
int main() {
    rabbit a;
    cin >> a.name >> a.weight >> a.puffyness >> a.colour;
    cout << a.name << " : " << a.weight << "kg, " << a.puffyness << " puffy, "
<< a.colour;
    …
}
```

Переопределение операторов ввода и вывода

```
struct rabbit {
    …
}
rabbit rabbit_in() {
        double w, p;
        string n, c;
        cin >> n >> w >> p >> c;
        return rabbit(w,p,n,c)
}
void rabbit_out(rabbit a) {
        cout << a.name << " : " << a.weight << "kg, " << a.puffyness << " puffy,
" << a.colour;
}
int main() {
    rabbit a;
    a = rabbit_in();
    rabbit_out(a);
    …
}
```

Переопределение операторов ввода и вывода

```
struct rabbit {
    …
    friend istream& operator>> (istream  &input, rabbit& a) {
        input >> a.name >> a.weight >> a.puffyness >> a.colour;
        return input;
    }
    friend ostream& operator<< (ostream &output, const rabbit& a) {
        output << a.name << " : " << a.weight << "kg, " << a.puffyness << "
puffy, " << a.colour;
        return output;
    }
}
int main() {
    rabbit a, b, c, d, e, f;
    cin >> a >> b >> c >> d >> e >> f;
    cout << a << '\n' << b << '\n' << c << '\n' << d << '\n' << e << '\n' << f;
    …
}
```

Переопределение операторов ввода и вывода

```
struct rabbit {
    …
}
int main() {
    rabbit a,b,c,d,e,f;
    cin >> a.name >> a.weight >> a.puffyness >> a.colour;
    cin >> b.name >> b.weight >> b.puffyness >> b.colour;
    cin >> c.name >> c.weight >> c.puffyness >> c.colour;
    cin >> d.name >> d.weight >> d.puffyness >> d.colour;
    cin >> e.name >> e.weight >> e.puffyness >> e.colour;
    cin >> f.name >> f.weight >> f.puffyness >> f.colour;
    cout << a.name << " : " << a.weight << "kg, " << a.puffyness << " puffy, "
<< a.colour << ‘\n’;
   bcout << b.name << " : " << b.weight << "kg, " << b.puffyness << " puffy, "
<< b.colour << ‘\n’;
    cout << c.name << " : " << c.weight << "kg, " << c.puffyness << " puffy, "
<< c.colour << ‘\n’;
    cout << d.name << " : " << d.weight << "kg, " << d.puffyness << " puffy, "
<< d.colour << ‘\n’;
    cout << e.name << " : " << e.weight << "kg, " << e.puffyness << " puffy, "
<< e.colour << ‘\n’;
    cout << f.name << " : " << f.weight << "kg, " << f.puffyness << " puffy, "
```

Заключение

Структуры, как правило, используются в решении задач на геометрию (для хранения координат точек, параметров однотипных фигур и т.д.), но могут использоваться и в других ситуациях, когда требуется чисто и быстро работать с объектами с несколькими параметрами.

Вопросы?