# **Лекция 1-2. Введение в язык программирования С++**

Организационные вопросы:

* Зачет выставляется по практическим работам
* Есть курсовой проект (составить программку, которая должна показать, чему мы научились)

Вопросы по лекции:

1. C++: составные части и концепции программирования

2. Данные и операции с данными

3. Управление программой

# **1. C++: составные части и концепции программирования**

Библиотека стандартных шаблонов

Средства объектно-ориентированного программирования

Язык программирования С

• Процедурное программирование

• Объектно-ориентированное программирование (будет сделан упор)

• Обобщенное программирование

С++ был создан Д. Ритчи, в конце 70х – начале 80х гг. язык был дополнен классовыми понятиями датчанином Б. Страуструпом. Завершил создание языка А. Степанов.

# **Литература**

* Шилдт Г. С+\* для начинающих. - СПб.: Питер, 2024.- 608 с.
* Шилдт Г. С++: Базовый курс. 3-е изд. Пер. с англ.- М.: Вильямс, 2019. - 624 c.
* Шилдт Г. С++: Методики программирования Шилдта. Пер. с англ.- М.: Вильямс, 2017. - 480 с.
* Шилдт Г. С++. Полное руководство. Классическое издание. Пер. с англ. - М.: Вильямс, 2020. - 800 с.
* Страуструп Б. Язык программирования С++. 3-е изд. Пер. с англ. - М.: Бином, 2022. - 1216 с.
* Павловская Т.А. С/С++. Программирование на языке высокого уровня. -СПб.:Питер, 2018.- 461 с.
* Рао С. С++ за 21 день. 8-е изд. Пер. с англ. - СПб.: Диалектика, 2020. - 752 с.

# **Процедурное программирование**

Программа объединяет в себе:

* данные;
* операции, проводимые с данными;
* управление выполнением операций.

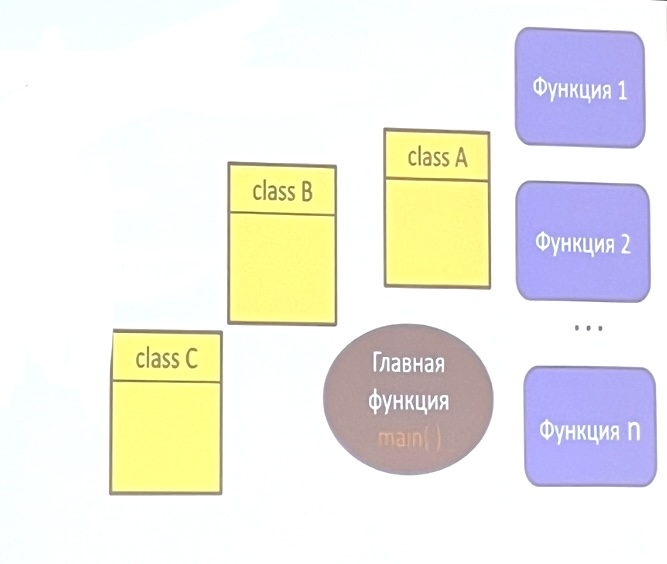
# **Структура программы:**

Главная функция – main ()

Другие функции

Программа – набор функций, который обязательно содержит главную функцию.

# **Объектно-ориентированное программирование**

Принципы и средства:

* инкапсуляция (классовость);
* наследование;
* полиморфизм;

# **Обобщенное программирование**

**Обобщённое программировани**е – это такой подход к разработке программного обеспечения, при котором проектирование данных и процедур их обработки производится безотносительно к конкретным типам данных, для которых этот алгоритм будет выполняться.

При программировании используются каркасы – универсальные структуры данных и алгоритмы их обработки – которые конкретизируются в программе реальными типами данных.

# **2. Данные и операции с данными**

Основные типы данных:

* **char** - символ;
* **int** -целое число;
* **float** - дробное (вещественное) число.

Тип данных определяет:

* внутреннее представление данных в памяти компьютера;
* размер памяти, отводимой под данный тип (множество значений, которые могут принимать величины этого типа);
* операции и функции, которые можно применять к данным этого типа.

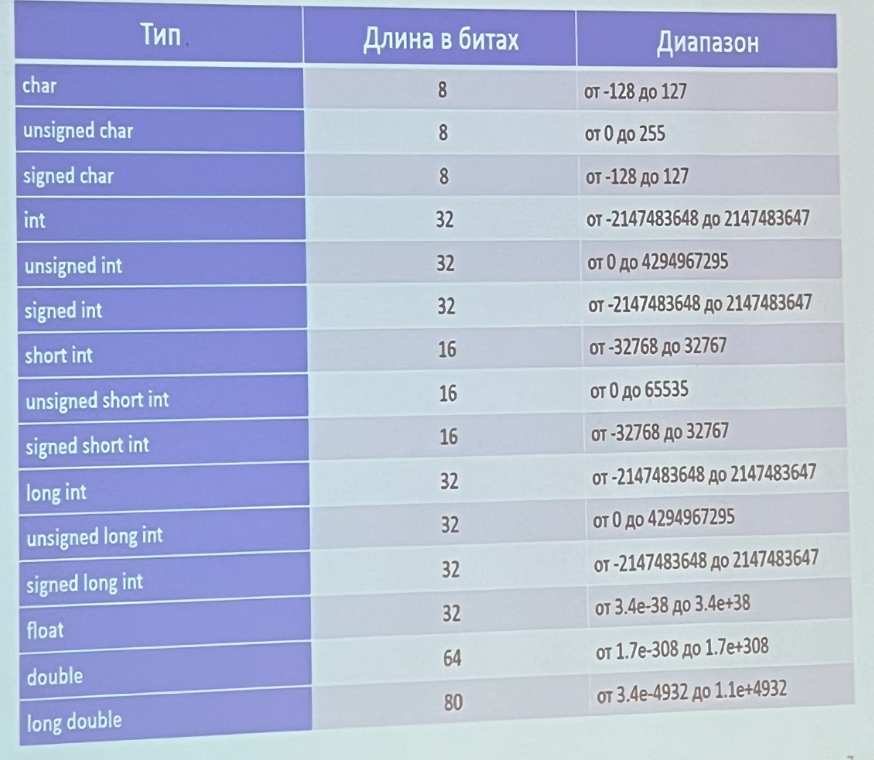
Модификаторы типа данных:

* **signed** (знаковое)
* **unsigned** (беззнаковое)
* **long** (длинное)
* **short** (короткое)

Особый тип данных - логический - **bool**, может принимать два значения:

* **true (истина);**
* **false (ложь).**

Все допустимые комбинации базовых типов и модификаторов для 32-разрядной среды приведены в таблице.



Стандартная библиотека использует тип double.

# **Переменные**

**Идентификаторы** – это имена, которые присваиваются переменным, функциям, операторам и т.д.

**Переменная** – это именованная область памяти, в которой хранятся данные определенного типа.

Объявление переменных.

**float a1;** // объявлена переменная а1 для дробных чисел

**char s;** // объявлена символьная переменная s.

**int a, b, c;** // объявлены три целочисленные переменные

**bool mark;** // объявлена переменная mark типа bool.

**Нотации** - способы написания идентификаторов

«Верблюжий стиль»: **firstNumber, flagMemory** (первая буква первого слова - в нижнем регистре).

Венгерская нотация: **iFirstNumber** (целочисленная переменная),

**bFlagMemory** (bool-переменная)

Содержимым переменных являются константы.

**int max = 768;**

# **Константы**

Целые: 37, -5, 47863, 3, 0, -374

Дробные: 2.73, 853.2, -12.542, 0.001

Символьные: 'a', 'G', 'A', 'j'

Строковые: "SPB", "Строка", "Frame"

Шестнадцатеричные: 0x5F, 0Х28А3

Восьмеричные: 012, 074632, 0144

# **Именованные константы**

Два варианта задания строковых констант:

1) Применение директивы **#define;**

**#define PI 3.1415**

**#define YEAR 2024**

2) Использование ключевого слова **const.**

**const float PI = 3.1415;**

**const int YEAR = 2024;**

# **Операции с данными**

**Арифметические операторы**

|  |  |  |
| --- | --- | --- |
| Сложение | + | c=a+b |
| Вычитание | - | d=a-b |
| Умножение | \* | s=a\*b |
| Деление | / | p=a/b |
| Деление по модулю (остаток от деления) | % | int a=19;  int b=8;  D=a%b; |

**Операторы сравнения**

**<, <=, >=, ==, !=.**

**Логические операторы**

логическое отрицание (инверсия): **!**

логическое «или» (дизьюнкция): **||**

логическое «и» (конъюнкция): **&&**

**Консольный ввод-вывод**

cin >> a;

cin >> k >> 1 >> m; //ввод с клавиатуры

/\* вывод на экран с переводом курсора на новую строку \*/

cout<<a<<"\n";

//, /\* - комментарии

# **Операторы инкремента (++) и декремента (--)**

Эти операторы увеличивают (++) или уменьшают (--) на единицу значение переменной, с которой они используются.

Эти операторы существуют в двух формах:

* в префиксной форме (++х) значение операнда изменяется прежде, чем оно используется в выражении;

**int x=3;**

**int y=++x;** // y=4, x=4

* в постфиксной форме (x++) предыдущее значение сначала используется в выражении и только потом значение операнда изменяется.

**int x=3;**

**int y=x--; //** y=3, x=2

**Пример:** каков будет результат?

**int a = 5;**

**int y = a+++a;**

**cout  << y;**

Insert 2 numbers: 24 35

Result: 59

Process returned 0 (0x0)

Press any key to continue.

Приоритет операторов: **+, --, -, \*, /, %, +, -**

унарный минус

**Первая программа**

#include < iostream>

using namespace std;

int main ()

{

int a;

int b;

int c;

cout << "Insert 2 numbers: "

cin >> a >> b;

c = a + b;

cout « "Result: "<< c <<'\n';

return 0;

}

# **Понятие массива**

****Массив относится к составному типу данных. Он представляет собой набор простых однотипных данных. Они расположены в памяти вплотную друг за другом.

**int m[100];**

**char s[25];**

**float drob[10];**

**int intVal[3]={70, 855, - 19};**

При использовании массива в программе число в скобках указывает *номер элемента массива* и называется *индексом*. Индекс всегда начинается с нуля.

#include<iostream>

using namespace std;

int main ()

{

char mass[6];

int k, 1, m;

cout<<"Insert 3 numbers (0-5)";

cin>>k>> 1>>m；

mass[0]='e';

mass[1]='0';

Insert 3 numbers (6-5): 0 2 5

end

Process returned e (8x8)

Press any key to continue.

mass[2]='n';

mass[3]='p'；

mass[4] ='r'；

mass[5]='d';

cout<<mass[k] <<mass[1]<<mass[m];

return 0;

}

# **Преобразование типов данных**

Преобразование значения переменной одного типа в значение другого типа называется приведение типа и бывает явным и неявным:

* при явном приведении перед выражением следует указать в круглых скобках имя типа, к которому необходимо преобразовать исходное значение;
* при неявном приведении преобразование происходит автоматически.

Пример явного приведения типа.

#includeiostream›

using namespace std;

int main ()

{

int x = 27;

double y = 5.83;

double z;

x = y; *// неявное приведение типа к int*

x=5

z=5

Process returned 0 (0x0)

Press any key to continue.

z = (int)y; *// явное приведение типа к int*

cout<< " x = "<< x <<'\n';

cout<<"z="<<z<<'\n';

return 0;

}

Преобразование значения переменной одного типа в значение другого типа называется приведение типа и бывает явным и неявным:

* при явном приведении перед выражением следует указать в круглых скобках имя типа, к которому необходимо преобразовать исходное значение;
* при неявном приведении преобразование происходит автоматически.

Пример явного приведения типа.

…

char c;

int d;

с = 'A';

d = c; //d=65

d = 67;

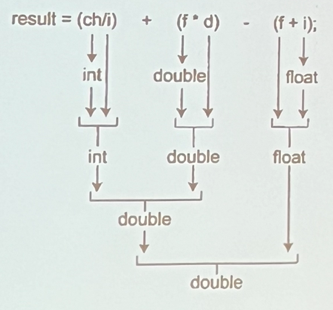
c = d; //c= 'C'

…

Если в выражении встречаются два операнда разных типов, то действуют следующие правила:

* все операнды преобразуются к типу наибольшего операнда. Процесс такого преобразования называется расширением типа;
* все типы char и short int преобразуются к типу int. Процесс такого преобразования называется целочисленным;
* если один из операндов имеет тип double, тогда любой другой операнд приводится к типу double. Даже, в случае с типом char, происходит приведение к типу double.

Пример.

char ch;

int i;

float f;

double d;

result=(ch/i)+(f\*d)-(f+i)

# **Оператор sizeof**

Размер памяти, который отводится под конкретный тип данных, в общем случае зависит от платформы компьютера. Чтобы определить, сколько байт памяти выделено под ту или иную переменную, в С++ используется оператор sizeof ().

#include ‹iostream>

INT: 4

LONG: 4

FLOAT: 4

DOUBLE: 8

LONG DOUBLE: 16

Process returned 0 (0x0)

using namespace std;

int main ()

{

int integer;

long longer;

float drob;

double db;

cout << " INT: "<< sizeof (integer) << endl;

cout « " LONG: "<< sizeof (longer) << endl;

cout « " FLOAT: "<< sizeof (drob) << endl;

cout « " DOUBLE: "<< sizeof(db) << endl;

cout « " LONG DOUBLE: "<< sizeof (long double) << endl;

return 0;

}

# **3. Управление программой**

**Ввод - вывод данных**

Система ввода-вывода С++ действует через потоки (streams). Поток ввода-вывода – это логическое устройство, которое выдает и принимает пользовательскую информацию.

Потоки сin и cout использовались в программах как стандартные средства обмена информацией с клавиатурой и монитором (с консолью).

int a=47; int num, b, iData;

cout<< a; cin >> num >> b >> iData;

cout<<5\*a+23；

**Форматирование данных**

Управляющие константы

|  |  |
| --- | --- |
| Изображение | Наименование |
| \a | Звуковой сигнал |
| \b | Возврат шага |
| \f | Перевод страницы (формата) |
| \n | Перевод строки |
| \r | Возврат каретки |
| \t | Горизонтальная табуляция |
| \v | Вертикальная табуляция |

**Манипуляторы** – это ключевые слова языка С++, которые применяется в выражениях ввода/вывода для управления информацией ввода/вывода.

Манипуляторы различают с параметрами и без параметров. Например:

|  |  |  |
| --- | --- | --- |
| Манипуляторы | Назначение | Ввод/вывод |
| dec | Ввод/вывод данных в десятичной форме | ввод и вывод |
| endl | Вывод символа новой строки с передачей в поток всех данных из буфера | вывод |
| ends | Вывод нулевого символа | вывод |
| flush | Передача в поток содержимого буфера | вывод |
| hex | Ввод/вывод данных в шестнадцатиричной системе | ввод и вывод |
| oct | Ввод/вывод данных в восьмеричной форме | ввод и вывод |
| resetiosflaqs(long f) | Сбрасывает флаги, указанные в f | ввод и вывод |
| setbase(int base) | Устанавливает базу счисления равной параметру base | вывод |
| setfill (int ch) | Устанавливает символ заполнения равным ch | вывод |
| setiosflags(long f) | Устанавливает флаги, указанные в f | ввод и вывод |
| setprecision(int p) | Устанавливает число цифр после запятой | вывод |
| setw (int w) | Устанавливает ширину поля равной w | вывод |
| ws | Пропускает начальный символ-разделитель | ввод |

#include<iostream>

#include<iomanip>

using namespace std;

int main ()

{

cout << hex << 100 << endl;

64

12

XXXXXXXX144Hello!

cout << oct <<10 << endl;

cout <<setfill('x') << setw (10);

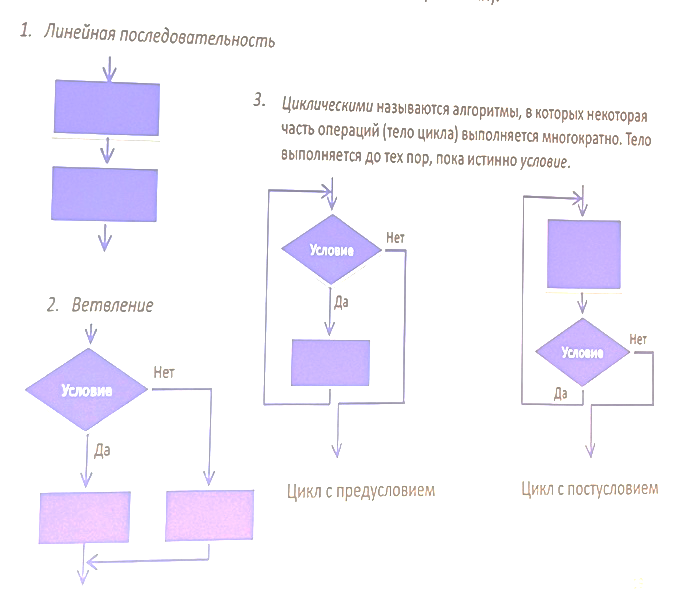
cout <<oct <<100<<" Hello!"<<endl;

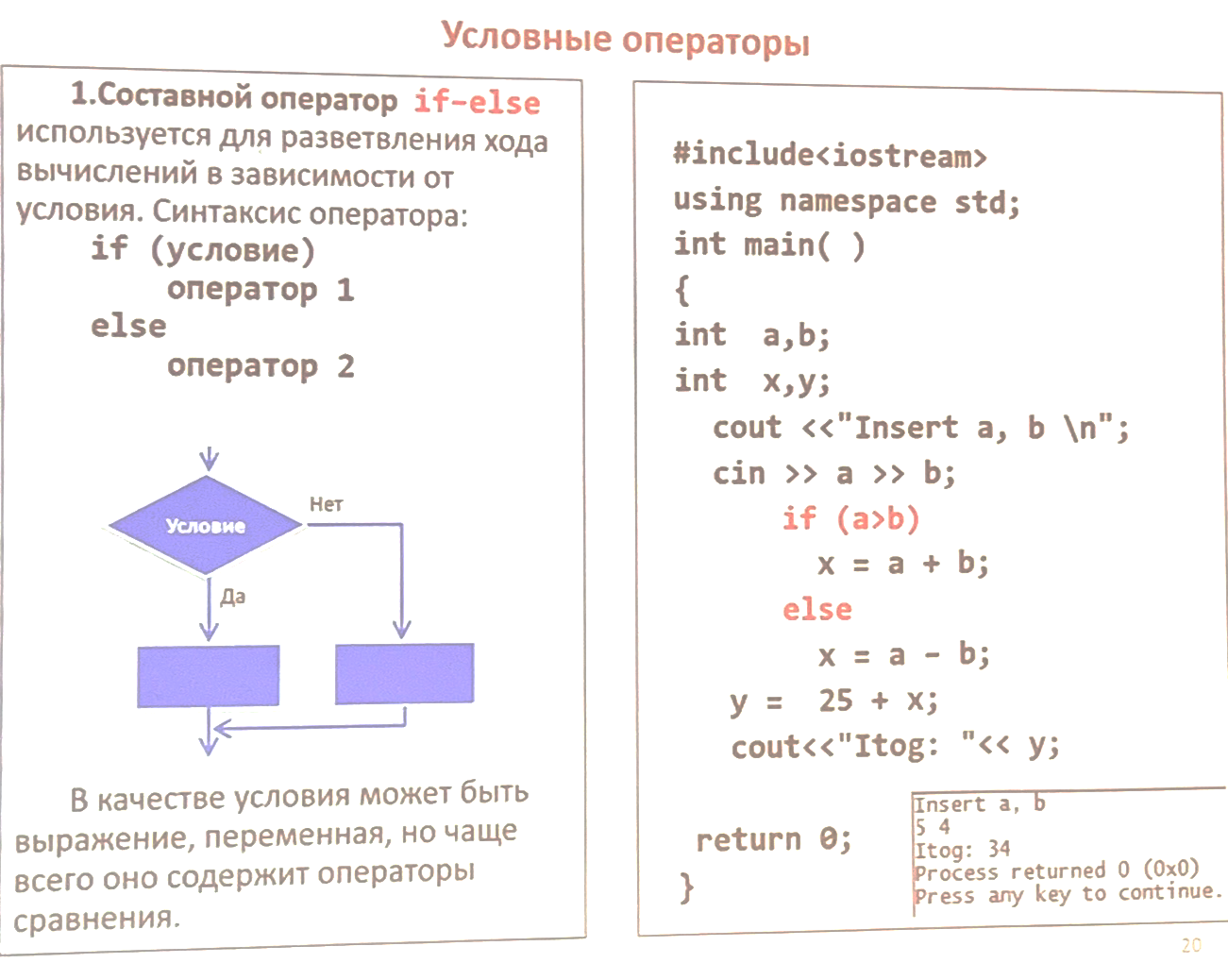
return 0;

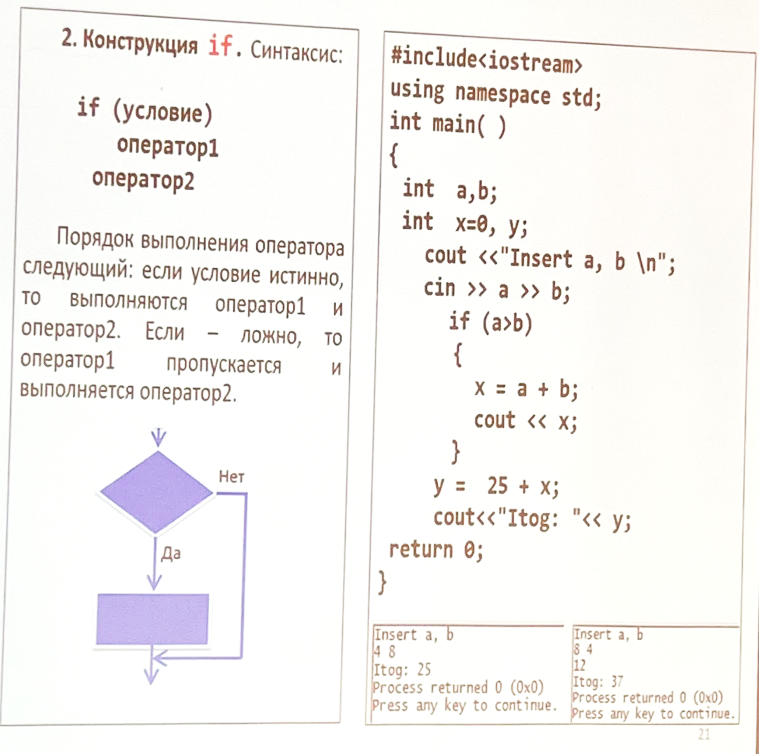
}

# **Базовые алгоритмические конструкции**

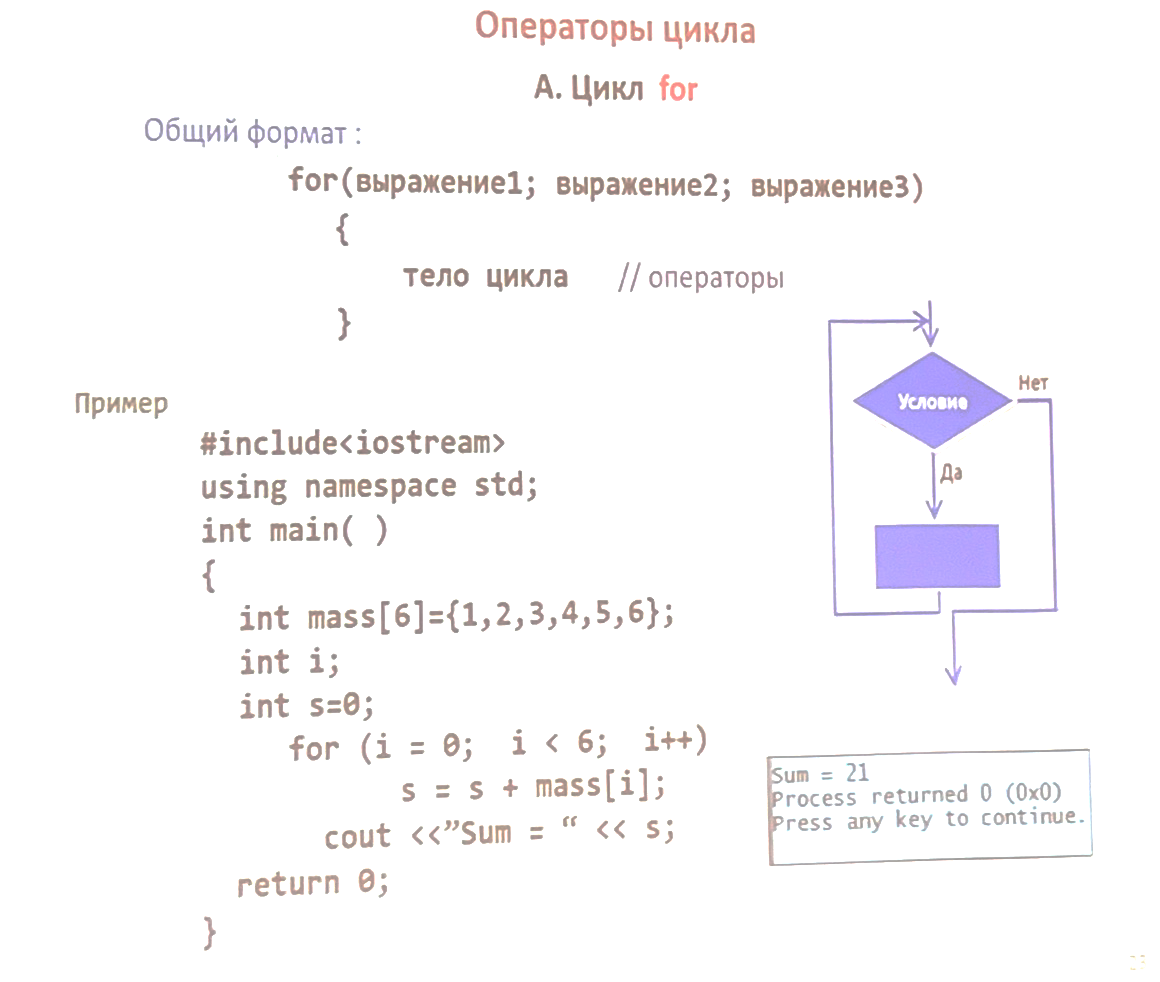
Любая программа с точи зрения последовательности выполнения команд может быть представлена тремя базовыми схемами (алгоритмами).

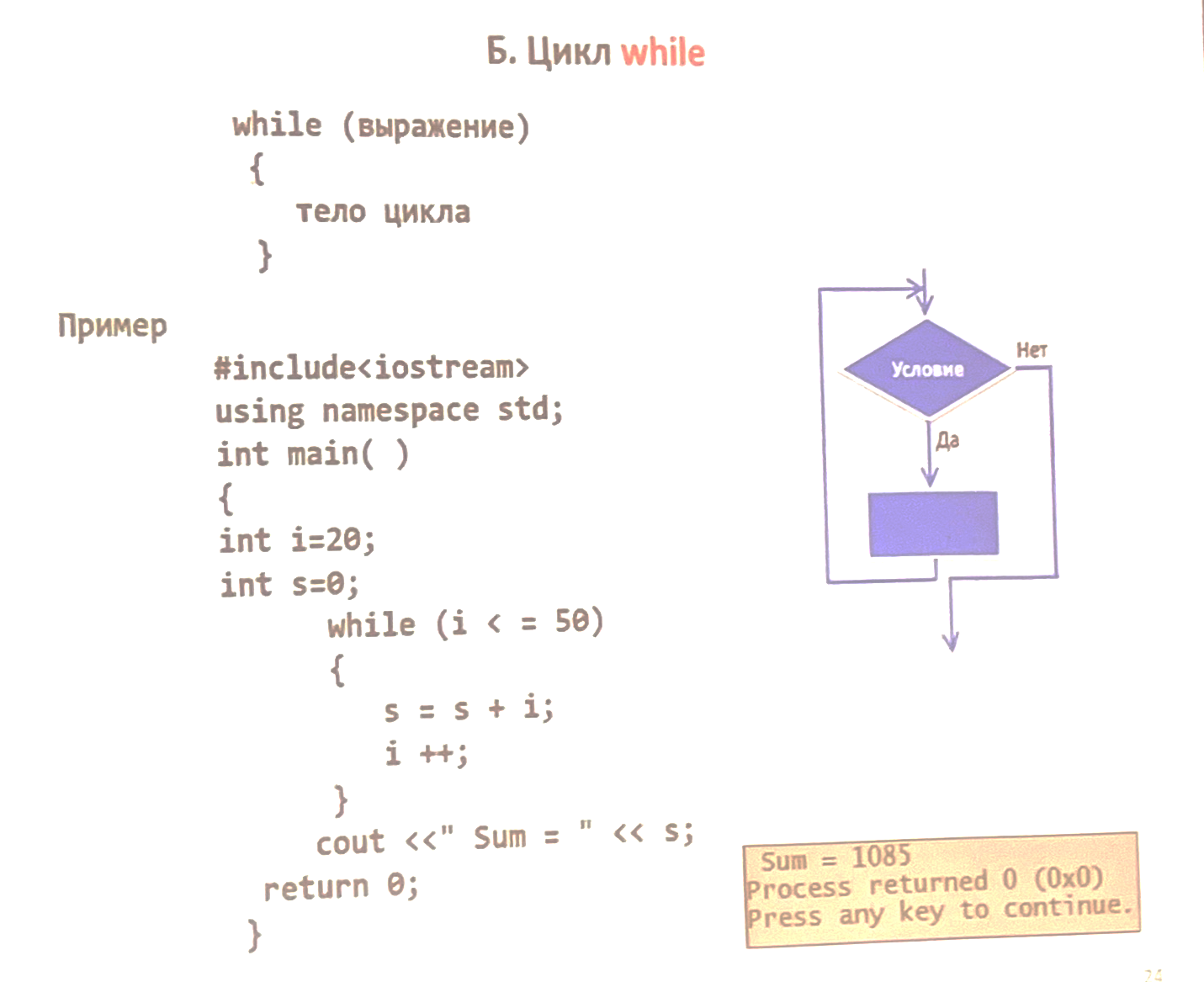


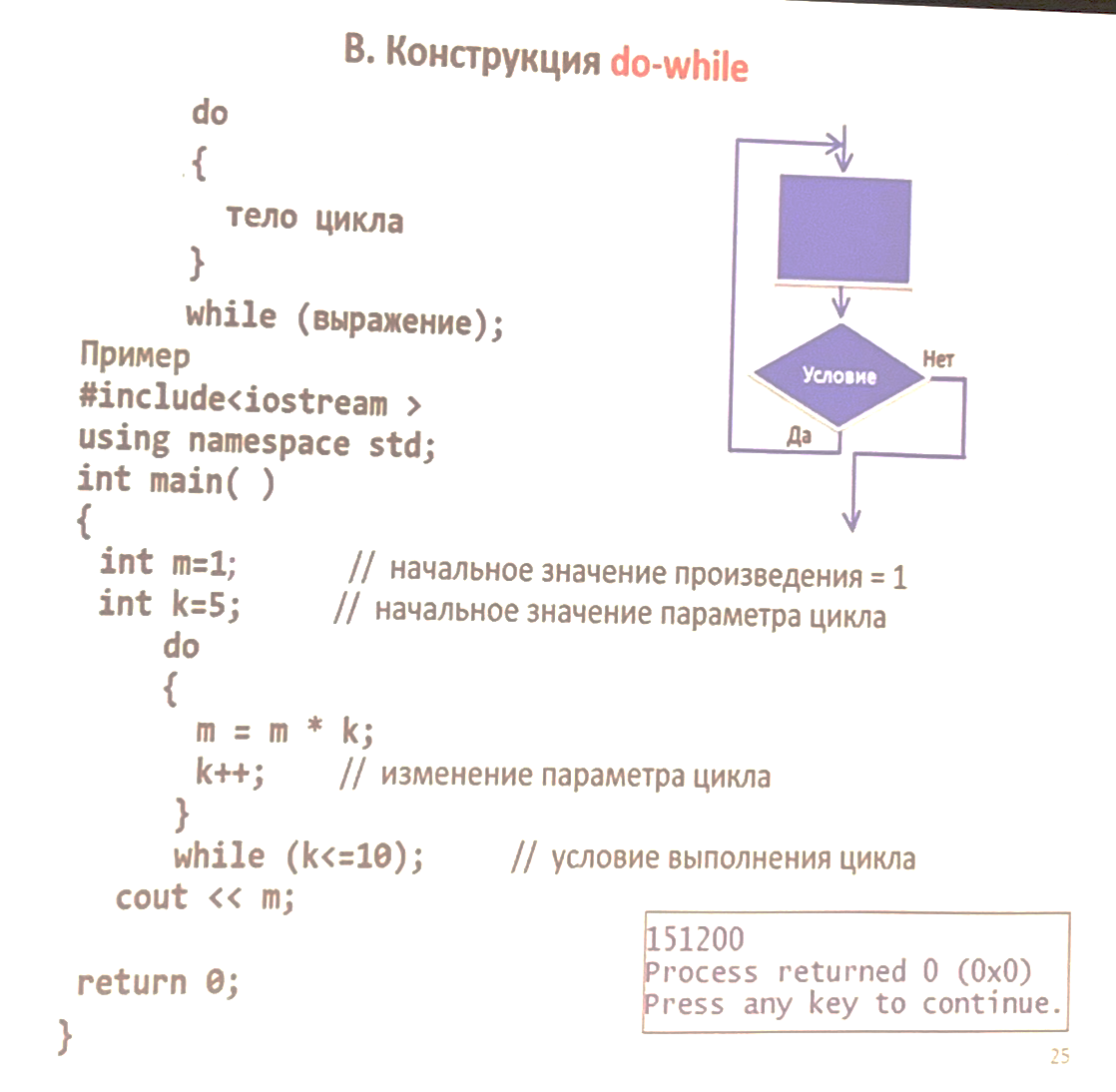












# **Лекция 3. Функции как самостоятельные части программы. Области видимости. Пространства имён.**

Вопросы:

1. Функции

2. Взаимодействие функций в программе

3. Внешние переменные и области видимости

4. Пространства имён  
**1. Функции**

Все функции разделяются на две части. Первую часть составляет большой набор библиотечных функций языка С\*. Например, основные математические функции определяются заголовком «cmath›, и подключаются к программе с помощью директивы #include<cmath›.

Вторая часть – это функции, которые разрабатывает сам программист и использует их в своих программах.

Функция – это логически самостоятельная именованная часть программы, в которую может передаваться любое количество значений аргументов, а

Функция может возвращать значение. Но только одно.

**Объявление функции**

При объявлении функции указывается (слева направо):

• тип возвращаемого функцией значения;

• имя функции;

• в круглых скобках - типы и имена параметров (переменных).

1. int buf (int a);
2. float funi (float b, char ch);
3. void beta (double m[10]);
4. char input (void);
5. char input ( );

**Определение функции**

Определение функции – это описание операций, которые выполняются в ее рамках.

Определение функции начинается заголовком, в котором указывается её прототип (тип возвращаемого значения, имя функции, список параметров), затем в фигурных скобках описываются действия, которые выполняются функцией (что называется телом функции). Если функция возвращает значение, то последним оператором в теле функции должен стоять оператор **return.**

int summa (int x, int y)

{

int z;

z=x+y;

return z;

}

int summa(int x, int y)

{

return x+y;  
}

**Вызов функции**

Для вызова функции надо указать:

1. ее имя и
2. в скобках - список аргументов в соответствии с прототипом.

summa (a, b);

function1(a, alpha, da);

w = delta (x,);

**2. Взаимодействие функций в программе**

#include‹iostream›

using namespace std;

int square(int a);//объявление функции square()

int main ()

#include<iostream> using namespace std;

void show(char x)

{

cout<< x << endl;

}

int main( )

{

char a = 's' ;

show(a);

return 0;

}

{

int z;

int x = 3;

z = square(x);

cout<<" square =" << z<<"\n";

return 0;

}

// определение функции square

int square (int a)

sum=25

Process returned 0 (0x0)

Press any key to continue

{

int b;

b = a\*a;

return b; // оператор возврата управления

}  
**Передача массива в функцию**

#include <iostream>

using namespace std;

void summa (float a[4])

{

float s=0; int i;

for (i=0; i<4; i++)

s=s+a[i];

cout <<s;

}

int main ()

{

float m[4];

Insert element: 1.05

Insert element: 2.5

Insert element: 3.005

Insert element: 4.0005

10.5555

Process returned 0 (0x0)

Press any key to continue.

int i;

for (i=0; i‹4; i++)

{

cout << "Insert element: ";

cin >>m [i];

}

summa (m);

return 0;

}

**3. Внешние переменные и области видимости**

Переменные, которые объявлены внутри функции, называются внутренними переменными или локальными. Каждая локальная переменная функции возникает только в момент обращения к этой функции и исчезает после выхода из нее. Такие переменные называются автоматическими, т. к. они образуются и исчезают одновременно с входом в функцию и выходом из нее; они не сохраняют своих значений от вызова к вызову.

#include <iostream>

using namespace std;

int summa (int a, int b)

{

return a+b;

}

int main()

{

Insert two int: 3 4

Summa = 7

Process returned 0 (0x0)

Press any key to continue.

int a, b;

cout <<" Insert two int: "; cin >> a >> b;

cout «" Summa = "« summa (a, b);

return 0;

}

Переменные а и b - локальные, поэтому однозначности не возникает, когда одни и те же имена переменных используются в различных функциях.

Чтобы локальная переменная могла сохранять свои значения от вызова к вызову функции, необходимо переменную отнести к классу

из функции.

памяти static (статические). Такие переменные не исчезают при выходе

#include<iostream>

using namespace std;

void par(int i)

{

static int p=21;

cout <<”i= ” <<i<<”p=”<<p<<endl;

p++;

}

int main( )

{

i=1 p = 21

i=2 p = 22

i=3 p = 23

Process returned 0 (0x0)

Press any key to continue.

Press any key to continue.

int i;

for (i=1; i<4; i++)

par (i);

return 0;

}

Если убрать **static:**

#include<iostream>

using namespace std;

void par(int i)

{

int p=21;

cout <<”i= ” <<i<<”p=”<<p<<endl;

p++;

}

int main( )

{

i=1 p = 21

i=2 p = 21

i=3 p = 21

Process returned 0 (0x0)

Press any key to continue.

Press any key to continue.

int i;

for (i=1; i<4; i++)

par (i);

return 0;

}

Чтобы переменная не только сохраняла свои значения в течение работы программы, но, и чтобы ею могли пользоваться другие функции, она должна быть не внутренней, а внешней (глобальной) переменной. Внешние переменные доступны повсеместно. Их можно использовать в любом месте программы, в любой функции. Кроме того, поскольку внешние переменные существуют постоянно, а не возникают и не исчезают на период выполнения функции, свои значения они сохраняют и после возврата из функций, их установивших.

Объявление внешних переменных производится вне функций в любом месте программы.

**Пример.**

#include<iostream>

using namespace std;

void square(); //объявление функции

int a, b; // объявление внешних переменных

int main ()

{

а = 3;

square();

int z = b;

sum=25

Process returned 0 (0x0)

Press any key to continue.

a = 4;

square();

z = z+b;

cout << " sum = " << z << endl;

return 0;

}

void square ( )

{

b=a\*a;

}

**Области видимости**

Область видимости переменной (функции) – это часть программы, в которой этой переменной можно пользоваться. Для локальных переменных – это функция, где они объявлены.

Область действия внешней переменной простирается от точки программы, где она объявлена, до конца файла.

void func1();

int func2 (int b);

int main()

{

…

}

int sp=0;

double val[10];

void func1( )

{

…

}

int func2 (int b)

{

…

}

**4. Пространства имен**

Пространства имен предназначены для локализации имен идентификаторов во избежание конфликтов имен.

В среде программирования С++ сосуществует огромное количество имен переменных, функций и классов. До введения понятия пространств имен все эти имена находились в одном глобальном пространстве имен, и возникало множество конфликтов. Конфликты имен имеют место, когда в одной и той же программе используются библиотеки функций и классов разных производителей. В этом случае вполне возможно - и даже очень вероятно, - что имена, определенные в одной библиотеке, будут конфликтовать с теми же именами, но определенными в другой библиотеке.

В ранних версиях языка вся библиотека С++ определялась в глобальном пространстве имен. Теперь библиотека С++ определяется в собственном пространстве имен **std**, что значительно снижает вероятность конфликтов имен.

Ключевое слово **namespace** путем объявления именованных областей дает возможность разделить глобальное пространство имен. По существу, пространство имен определяет область видимости.

Обозначение пространства имён с помощью ключевого слова **namespace**:

**namespace имя {**

**// объявления**

**}**

Все, что определено внутри инструкции namespace, находится внутри области видимости данного пространства имен.

Пример объявления пространства имен **MyNameSpace:**

**namespace MyNameSpace {**

**int i, k;**

**void myfunc (int j) { cout « j; }**

**void seti (int x) { i = x; }**

**}**

Здесь мена переменных і и к, функции **myfunc()** и **seti()** находятся в области видимости, определенной пространством имен **MyNameSpace.**

К идентификаторам, объявленным в пространстве имен, внутри этого пространства можно обращаться напрямую. При обращении извне пространства имен к идентификаторам, объявленным внутри этого пространства, следует указывать оператор расширения области видимости.

**MyNameSpace:: i = 10;**

**MyNameSpace:: seti(287);** //вызов функции seti()

Если в программе обращения к членам пространства имен происходят достаточно часто, то следует использовать инструкцию **using**.

У этой инструкции имеются две основные формы:

**using namespace имя;**

**using имя:: представитель;**

**using MyNameSpace:: k;** // видимой делается только переменная к

**k = 210;**  // инструкция правильна, поскольку переменная к видима

**using namespace MyNameSpace;** //видимыми делаются все члены пространства имен MyNameSpace

**i = 215;** // инструкция правильна, поскольку видимы все члены

пространства имен MyNameSpace

Стандарт Standard C++ определяет целую библиотеку в собственном пространстве имен **std**. Именно по этой причине во всех программах должна указываться следующая инструкция:

**using namespace std;**

Эта инструкция делает пространство имен std текущим, что позволяет получить прямой доступ к именам функций и классов, определенных в библиотеке языка Standard C++, без необходимости каждый раз с помощью оператора расширения области видимости уточнять, что используется пространство имен **std**.

**Пример.**

#include<iostream>

using namespace std;

namespace func { //обозначение пространства имён

int square (int a)

{

return a\*a;

}

Square = 9

Process returned 0 (0x0)

Press any key to continue.

} // граница пространства имён

int main ()

{

int x = 3;

int z;

z = func:: square(x);

cout <<" Square = " << z << endl;

return 0;

}

# **Лекция 4. Указатели и массивы. Ввод-вывод данных**

Вопросы:

1. Понятие указателя

2. Использование указателей для связи между

Функциями

3. Связь указателей с массивами

4. Многомерные массивы

5. Файловый ввод / вывод

**1. Понятие указателя**

Указатель – это особый тип данных, предназначенный для хранения адресов, в которых находятся другие данные. Так как он указывает на данные, то его обозначение (\*) совмещается с типом данных, на которые он ссылается. Например:

**int\*** - указатель на данные типа **int**;

**double\*** - указатель на данные типа **double**.

Переменные-указатели также (для сокращения) называются указателями и объявляются по общим правилам. Например:

**int\* p;** // **р** - указатель на переменную типа **int**

**double\* sys;** // **sys** - указатель на тип **double**

Если имеется переменная, например **int beta=812;** то для получения её адреса используется унарный оператор **&** (амперсанд):

**p= &beta;** // р - указатель на переменную beta, т. к. данной

// операцией (&) ему присвоен адрес этой переменной

Для обращения к переменной через ее указатель используется оператор косвенного доступа - «\*» (обращение по адресу). Запись

**x = \*p;**

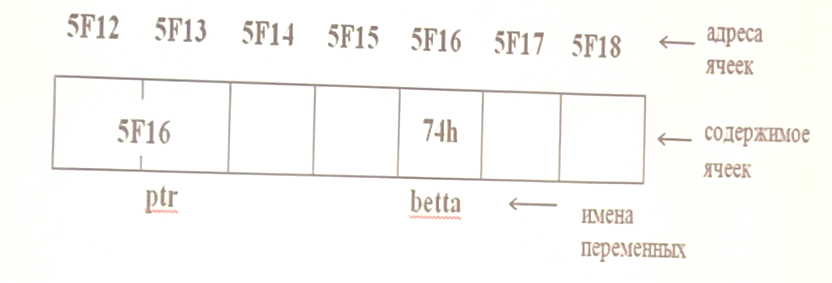
означает, что переменной х присваивается значение, которое хранится по адресу **р.**

**Связь переменной с её указателем**

char \*ptr; char betta;

betta = 't';

ptr = &betta;



Так как переменной **betta** присвоен символ t, то согласно кодовой таблице (например, ASCIl) ему соответствует код **74h** (в шестнадцатеричной системе счисления), что и будет записано в ячейку **5F16** (адреса ячеек обычно записываются в шестнадцатеричной системе счисления). А переменная **ptr** – это указатель на **betta**, поэтому он содержит адрес переменной betta, т.е. **5F16.**

После того, как указатель объявлен, он может использоваться в выражениях.

Например:

**\*ptr = 80;** // по адресу ptr будет записано 80

**\*p= \*p+10;**  // \*р увеличивается на 10

**(\*p)++;** //увеличивается на 1 содержимое по адресу **р**.

С указателями можно выполнять следующие операции: разадресация, или косвенный доступ (\*), присваивание, сложение с константой, вычитание, инкремент (+), декремент (--), сравнение.

Так, если int \*p; p=р+1;

то после выполнения этого фрагмента р будет указывать на следующий элемент типа **int**. Это означает, что компилятор будет **масштабировать** приращения в зависимости от типа данных, на который ссылается указатель.

#include <iostream>

using namespace std;

int main()

{

0x22ff08

0x22ff10

Process returned 0 (0x0) execution time: 0.486 s

Press any key to continue.

double var;

double\* ptr = &var;

cout << ptr << endl;

ptr++;

cout << ptr << endl;

return 0;

}

INI:4

LONG:4

FLOAT: 4

POINTER: 4

LONG DOUBLE: 12

Process returned 0 (0x0)

execution time: 0.514 s

Press any key to continue.

**Пример с размерами данных**

#include < iostream>

using namespace std;

int main ()

{

int integer;

long longer;

long\* pl = &longer;

float drob; cout < "INT:"« sizeof (integer) «endl;

cout « "LONG: "« sizeof (longer) «endl;

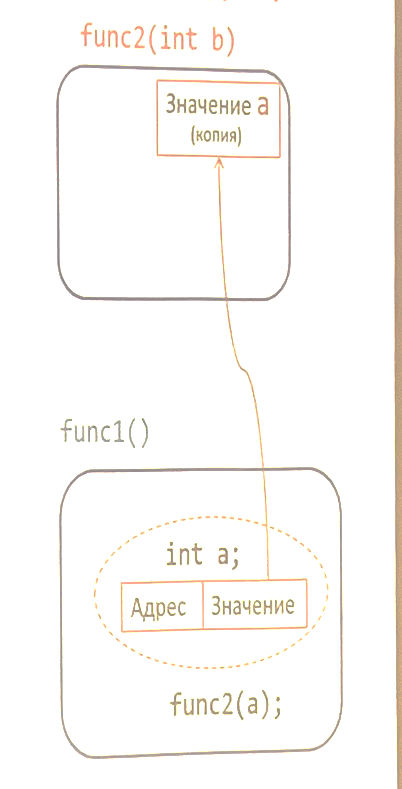
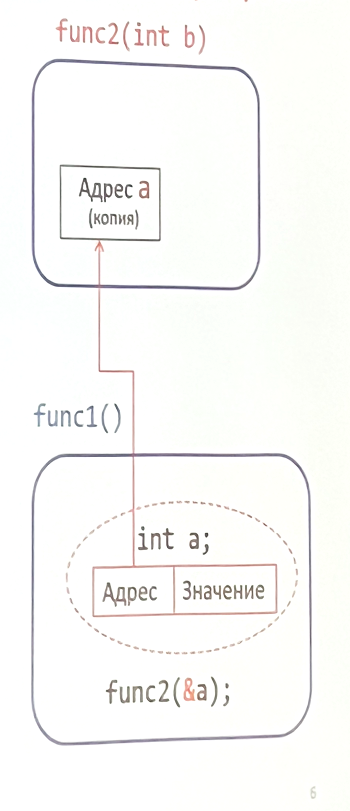
cout<<"FLOAT:"<<sizeof (drob)<<endl;

cout<<"POINTER:"<<sizeof(pl) <<endl;

cout << "LONG DOUBLE: "<<sizeof(long double) «<endl;

return 0;

}  
**2. Использование указателей для связи между функциями**



При передаче в функцию аргумента по значению:

func2(a);

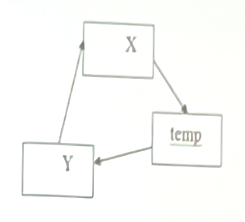
создается его копия, которая и используется в функции. Оригинал значения остается в вызывающей функции func1() и не подвергается изменению.

При передаче в функцию аргумента по адресу:

func2(&a);

создается копия адреса, где находится значение. Но этот адрес указывает на значение в вызывающей функции, т.е. на оригинал значения аргумента. Копия значения при вызове функции не создается и вызываемая функция func2() использует значение (оригинал) вызывающей функции func1().

**Пример**

Допустим, необходимо разработать функцию, которая меняет местами два элемента. Чтобы поменять местами значения х и у, следует ввести дополнительную переменную, например **temp**. На рисунке показан механизм, выполняющий данную задачу.

Чтобы получить желаемый эффект, вызывающей программе надо передать указатели на переменные, значения которых должны быть изменены. Поэтому при вызове функции, ей в качестве аргументов передаются **адреса переменных:**

swap (&a, &b);

Так как оператор **&** получает адрес переменной, то **&а** есть адрес переменной а.

В самой же функции **swap()** параметры должны быть объявлены как указатели.

void swap (int \*px, int \*py)

{

int temp;

temp = \*px;

\*рх = \*ру;

\*py = temp;

}

void swap (int x, int y)

{

int temp;

temp = x;

x = y;

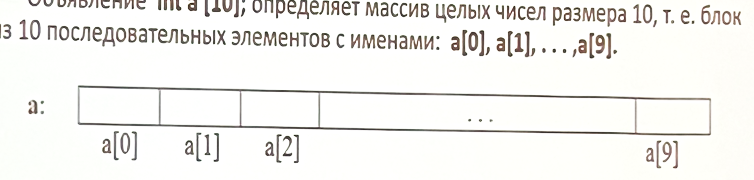
y = temp;

}

**3. Связь указателей с массивами**

Массив – это набор элементов (переменных, констант) одного и того же типа, расположенных в памяти вплотную друг за другом.

Объявление int a [10); определяет массив целых чисел размера 10, т. е. блок из 10 последовательных элементов с именами: а(0], а(2), ...,а (9).



Число в квадратных скобках указывает относительную позицию элемента (порядковый номер). Это число называется индексом. Индексация всегда начинается с нуля.

Если **int \*pa;** то в результате присваивания

**pa = &a[0];**

**ра** будет указывать на нулевой элемент массива, т. е. **ра** будет содержать адрес элемента **а[0].**

Если сделать запись х = \*ра;

то это означает, что переменной х присвоится значение а[0].

Если ра указывает на нулевой элемент массива, то **ра+1** будет указывать на первый элемент, а **ра + і** - на і- тый элемент. Таким образом **\*(ра + і)** есть

содержимое **а[i].**

Важное свойство организации массива в языке С++ заключается в том, что имя массива является указателем на его первый элемент.

Если

int a[10]; и pa = &a[0];

то и

a⬄ &a[0].

Таким образом, **а** и **&а[0]** – это одно и то же, а именно – адрес начального элемента массива. Поэтому любой элемент массива, например пятый, можно

представить в виде **\*(а + 5**). В общем случае записи a[i] и **\*(а + і)**

эквивалентны. Также будут эквивалентны и следующие записи **&a [i]** и **а + і.**

Между именем массива и указателем, выступающим в роли имени массива, существует одно различие. Указатель – это переменная, поэтому можно написать **ра = а** или **ра++.** Но имя массива переменной не является, оно = const. Поэтому записи вроде **а = ра** или **а++** недопустимы.

# **Лекция 5. Символьные строки**

Вопросы:

1. Определение символьных строк в программах

2. Консольный ввод - вывод строк

3. Передача символьных строк между функциями

4. Ввод-вывод символьных строк в файлы

5. Операции со строками

6. Тип данных «строка»

**1.Определение символьных строк в программах**

В С++ нет встроенного (простого) типа данных «строка». Символьные строки представляются массивом символов, в конце которого ставится пуль-символ – «\0».

Под определением строки понимается указание её символьного значения при объявлении. При определении строк символы строки заключаются в двойные кавычки.

Существует четыре способа определения строк:

• строковые константы;

• массивы символов;

• указатели на строки;

• массивы символьных строк.

**1. Строковые константы.**

Как только компилятор встречает нечто, заключенное в двойные кавычки, он определяет это как строку и размещает ее символы последовательно в памяти, добавляя в конце '\0'.

Для определения констант используется директива **#define**.

**Пример**

#define STR "Это строковая константа"

cout <<" Пример строки: "<<STR;

Когда в самой символьной строке необходимо поставить двойные кавычки, используется символ '\':

#define SIG "Принят сигнал \"Тревога\" "

**2. Массивы символов.**

char str [] = "Слово";

char str [6] = "Слово";

char str [6] = {'C', 'л', 'o', 'в', 'o', '\0'}:

В качестве напоминания следует отметить, что имя массива есть указатель (константа) на нулевой элемент массива: &str [0]<=>str, т.е. это – адрес нулевого элемента. Обращение по этому адресу дает букву 'С.

Для того чтобы вывести символьную строку на экран, достаточно в

операторе вывода указать только имя символьного массива:

cout << str;

**3. Указатели на символьные строки.**

char\*pst = "еще строка";

Встретив такую запись, компилятор создает указанную строковую константу **еще строка**, а также переменную-указатель **pst**, которой присваивает адрес первого элемента строки. Для вывода на экран используется имя указателя:

cout << pst;

**4. Массивы символьных строк.**

Этот способ используется для задания сразу нескольких символьных строк и применяется для хранения сообщений, выводимых в программе.

Пример:

char \* ps[] = {"строка1", "строка2", "строка3"};

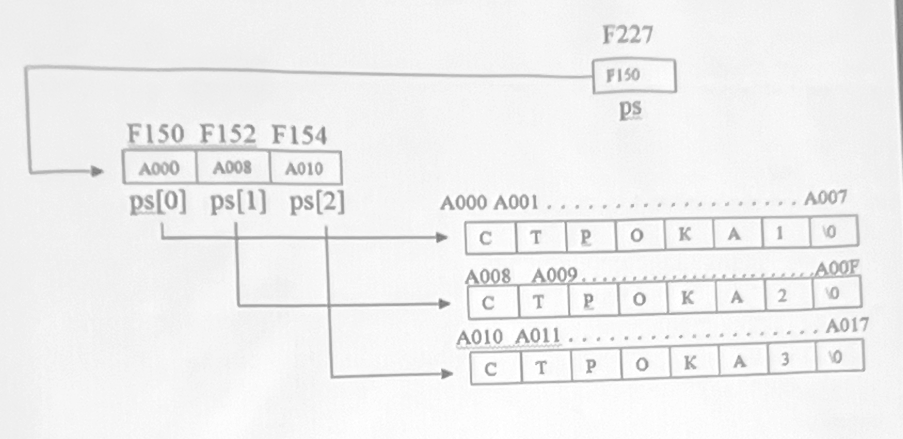
Встретив такую запись, компилятор создает:

а) переменную-указатель массива **ps**;

б) три переменные - указатели строк: **ps[0], ps[1], ps[2];**

в) три строковые константы: **строка1, строка2, строка3**.

char \* ps[] = {"строка1", "строка2", "строка3"};



Чтобы обратиться к первой строке, можно использовать запись **ps[0]** или **\*ps**. Обратиться к первому элементу первой строки можно **ps[0] [0]**, или **\*ps[0]**, или **\*\*ps**. Аналогично и для других строк.

**Пример**

#include <iostream>

using namespace std;

int main ()

{

setlocale (0,"");

int i;

char \*colour [3];

colour [0] - "синий";

colour [1] = "красный";

colour [2] = "зеленый";

cout<< "Введи цифру (0-2)"<<end1;

cin >> i;

if (i<0 || i>2)

cout＜ “неверный ввод"；

else

cout << ”Цвет-”<<colour[i];

return 0;

}

**2. Консольный ввод - вывод строк**

Использование символьного массива.

#include <iostream>

using namespace std;

int main()

Insert Seformation of symbols:

St. Petersburg

St. Petersburg

Process returned 0 (010)

Press any key to continue

{

char sps [28];

cout « "Insert information of symbols: "<< endl;

cin >> sps; // ввод с клавиатуры

cout << sps; // вывод на экран

return 0;

}

#include <iostream>

#include <cstdio>

using namespace std;

int main()

{ setlocale(0,””);

char str[20];

cout<<”Введите информацию: ”;

gets(str);

puts(str);

return 0;

}

Функция ввода строк:

gets (char \*string),

где **string** задает область памяти, в которую помещаются символы вводимой строки и откуда они выводятся на экран.

Для вывода на экран используется функция

puts (char \*string).

**3. Передача символьных строк между функциями**

#include <iostream>

using namespace std;

void get\_array(char s[]) /\* параметром является символьный

массив\*/

{

s[0]= 'b' ; //изменили первый символ

cout<<s<<endl;

}

int main()

{

char str[]="kasta"; // Инициализируем строку

get\_array(str); // Передали массив в функцию

return 0;

basta

Process returned 0 (0x0)

Press any key to continue.

}

// В этом примере строка обозначается через указатель

#include <iostream>

using namespace std;

void get\_array(char\*s)

{

s [0]='b'; //изменили первый символ

int main()

{

char str[]="aaaaa"; //Инициализируем строку

get\_array(str); //Передали массив в функцию

cout<<str<<endl; /\*Вывод на экран измененного массива\*/

return 0;

}

baaaa

Process returned 0 (0x0)

Press any key to continue.

// В этом примере строка возвращается из функции по указателю

#include < iostream>

using namespace std;

char\* get\_st()

{

char\*s ="Only string"; // Определение строки

return s;

}

int main()

{

char\* str;

str = get\_st( ); // Получение указателя на строку

cout << str << endl; // Вывод строки на экран

return 0;

Only string

Process returned 0 (0x0)

Press any key to continue.

}