1) Зачем нужен оператор if, оператор if else. Конструкция if else if else?

Оператор if (англ. “если”) служит для того, чтобы выполнить какую-либо операцию в том случае, когда условие (выражение/условное выражение) является верным.

Оператор **if** говорит нам, что если условие истинно, оно выполнит блок операторов, а если условие ложно, то не будет. Но что, если мы хотим сделать что-то еще, если условие ложно? Здесь идет оператор C / C ++ **else** (англ. “иначе”). Мы можем использовать оператор **else** с оператором **if** для выполнения блока кода, когда условие ложно

Конструкция оператора ***if***:

```
if (условие)
        инструкция_1;

if (условие) {
        инструкция_1;
        инструкция_2;
        }
```

Конструкция оператора ***if else***:

```
if (условие)
        инструкция_1;
else
        инструкция_2;

if (условие) {
        инструкция_1;
        инструкция_2;
        }
else {
        инструкция_3;
        инструкция_4;
        }
```

2) Перечислите операторы управления?

Операторы управления:

- Остановка (halt);
- Условное ветвление;
- Циклы;
- Прыжок (jump);
- Исключения.

3) Остановка (halt)?

Остановка (halt) — оператор управления порядком выполнения, который заставляет программу

немедленно прекратить выполнение.

4) Прыжок (jump)? 5) Оператор goto?

Прыжок (jump) оператор goto — это оператор управления порядком выполнения кода, который заставляет ЦП сделать переход из одного участка кода на другой, осуществить так называемый прыжок.

6) Оператор switch-case?

Управление потоком выполнения программы — условный оператор ветвления **switch-case**.

Конструкция **switch-case**:

switch (<Идентификатор/Выражение>) {

case (значение1):

инструкция1;

break;

case значение2:

инструкция2;

break;

...

default:

инструкция если ни один вариант не подошел;

break;

}

7) Лейбл по умолчанию (default case)?

После выражения **switch** мы объявляем блок. Внутри блока мы используем лейблы (англ. «***labels***») для определения всех значений, которые мы хотим проверять на соответствие выражению.

Лейбл по умолчанию (***default case***). Объявляется с использованием ключевого слова default. Код под этим лейблом выполняется, если ни один из кейсов не соответствует выражению switch. Лейбл по умолчанию является необязательным. В одном switch может быть только один default.

8) switch и fall-through?

switch - это ключевое слово, за которым следует выражение, с которым мы хотим

работать. Обычно это выражение представляет собой только одну переменную, но это может

быть и нечто более сложное, например, nX + 2 или nX − nY. Единственное ограничение к

этому выражению — оно должно быть интегрального типа данных (т.е. типа char, short, int,

long, long long или enum).

9) Объявление переменной и её инициализация внутри case?

Можем использовать несколько стейтментов под каждым кейсом, не определяя новый блок.

Можем объявлять, но не инициализировать переменные внутри блока case. В блоке switch все переменные, которые были объявлены до кейса case, будут доступны в этом кейсе.

Правило: Если нужно инициализировать и/или объявить переменные внутри кейса — используйте блоки стейтментов.

Нежелательно использовать присвоение значений переменным (плохо читается, может запутать) и вызов функции внутри switch, но вне кейсов.

10) Что такое ссылка.

Ссылка — это тип переменной в C++, который работает как псевдоним другого объекта или значения.

11) Какие в C++ поддерживает типы ссылок.

C++ поддерживает три типа ссылок:

1) ссылки на не константные значения (обычно их называют просто «ссылки» или «не константные ссылки»).

2) ссылки на константные значения (обычно их называют «константные ссылки»).

3) ссылки на r-value.

12) Инициализация ссылок на не константные типы.

Ссылка (на не константное значение) объявляется с использованием амперсанда (&) между типом и именем ссылки.

Конструкция:

<тип данных> &Идентификатор = Значение/Объект;

13) Инициализация ссылок на константы.

В отличие от ссылок на неконстантные значения, которые могут быть инициализированы только неконстантными l-values, ссылки на константные значения могут быть инициализированы неконстантными l-values, константными l-values и r-values.

14) Ссылки в качестве параметров в функциях.

Для передачи переменной по ссылке – нужно просто объявить параметры функции как ссылки, а не как обычные переменные. При вызове функции переменная станет ссылкой на аргумент.

15) Ссылки на массив и передача в функцию.

Передача массива в функцию представляет передача массива по ссылке. Прототип функции, которая принимает массив по ссылке, выглядит следующим образом:

Конструкция:

void print(int (&)[]);

Обратите внимание на скобки в записи (&). Они указывают именно на то, что массив передается по ссылке.

16) Когда необходимо использовать ссылки.

- при передаче структур или классов (используйте const, если нужно только для чтения);
- когда нужно, чтобы функция изменяла значение аргумента.

17) Что такое указатели?

Указатели — это переменные, которые содержат адреса памяти. Их можно разыменовать с помощью оператора разыменования (*) для извлечения значений из адресов, которые они хранят.

18) Указатели на типы данных. Объяснить что такое и привести примеры?

Указатели объявляются точно так же, как и обычные переменные, только со звездочкой между типом данных и именем переменной.

Конструкция:

<тип данных> *(идентификатор);

Синтаксически C++ принимает объявление указателя, когда звездочка находится рядом с типом данных, с именем переменной или даже посередине. Оператор (*) звездочка не является оператором разыменования. Это всего лишь часть синтаксиса объявления указателя. Однако при объявлении нескольких указателей звездочка должна находиться возле каждой переменной.

19) Указатели на функции. Объяснить что такое и привести примеры?

Возврат по адресу — это возврат адреса переменной обратно в caller. Подобно передаче по адресу, возврат по адресу может возвращать только адрес переменной. Литералы и выражения возвращать нельзя, так как они не имеют адресов. Поскольку при возврате по адресу просто копируется адрес из функции в caller, то этот процесс также очень быстрый.

Пример:

```
#include <iostream>
int g_value = 2;
int* Func(int x) {
g_value += 1;
```

```
return &g_value;
}
int main() {
int x = 2;
int rez = 0;
rez = *Func(x);
std::cout << "rez: " << rez << "\n";
std::cout << "addr g_value: " << &g_value << " g_value: " << g_value << "\n";
int *pint = Func(x);
std::cout << "addr Func: " << Func(x) << " value Func: " << *Func(x) << "\n";
std::cout << "addr pint: " << pint << " value pint: " << *pint << "\n";
}
```

20) Указатели на массивы. Объяснить что такое и привести примеры?

Распространенная ошибка думать, что переменная array и указатель на array — идентичны. Это не так. Хотя оба указывают на первый элемент в массиве, информация о типе у них разная. В примере выше тип array— int[4], тогда как тип указателя на массив — int *.

Путаница в основном вызвана тем, что во многих случаях, при вычислении, фиксированный массив распадается (неявно преобразовывается) в указатель на первый элемент массива. Доступ к элементам по-прежнему осуществляется через указатель, но информация, полученная из типа массива (например, его размер), не может быть доступна из типа указателя.

Основное различие возникает при использовании оператора sizeof(). При использовании в фиксированном массиве sizeof возвращает размер всего массива (длина массива * размер элемента). При использовании с указателем sizeof возвращает размер адреса памяти (в байтах). Например:

```
#include <iostream>
int main() {
int array[4] = { 5, 8, 6, 4 };
std::cout << sizeof(array) << '\n'; // выведется sizeof(int) * длина array
int *pa = array;
std::cout << sizeof(pa) << '\n'; // выведется размер указателя
return 0;
}
```

21) Нулевой указатель?

Нулевое значение (значение **null**) — это специальное значение, которое означает, что указатель ни на что не указывает. Указатель, содержащий значение null, называется нулевым указателем.

22) Указатель на указатель?

Указатель в C — не семантика, а механизм. Он сам по себе не несёт смысла, но может *использоваться* для выражения того или иного смысла. То же относится и к двойному указателю: он может использоваться для разных вещей.

23) Указатель на void?

Указатель типа void (или еще «общий указатель») — это специальный тип указателя, который может указывать на объекты любого типа данных!

Конструкция:

<void> *(идентификатор);

Объявляется он как обычный указатель, только вместо типа данных используется ключевое слово void.

24) Какие значения мы получим в результате выполнения следующей программы:

```
#include <iostream>
int main() {
short value = 13;
short value_1 = 100;
short *ptr = &value;
std::cout << &value << '\n';
std::cout << (value +=1) << '\n';
*ptr = 9;
std::cout << (value = value_1 + *ptr) << '\n';
std::cout << "Результат: " << value << '\n';
return 0;
}
```

Вывод адреса по ссылке: 0x61fe86
Вывод суммы значения переменной и 1: 14

Вывод суммы значения переменной и значения по указателю 109

Вывод значения: Результат: 109

Что не так со следующим фрагментом кода:

```
int main() {
int value = 45;
int *ptr = &value;
*ptr = &value; // указателю уже присвоили значение
ptr = value; // указателю уже присвоили значение
}
```

Что выведет программа:

```
#include <cstdlib> // нужно для функции exit()
#include <iostream>
int main() {
std::cout << "Hi !\n";
exit(0); // завершает выполнение программы и возвращает операционной системе 0
// Следующие стейтменты никогда не выполняться
std::cout << 3;
}
```

Программа выведет строку: Hi ! с переводом каретки на новую строку
Что выведет программа:

1)

```
#include <iostream>
int main() {
switch (2) {
case 1: // Не совпадает!
std::cout << 1 << '\n'; // пропускается
case 2: // Совпало!
std::cout << 2 << '\n'; // выполнение кода начинается здесь
case 3:
std::cout << 3 << '\n'; // это также выполнится
case 4:
```

```
std::cout << 4 << '\n'; // и это
default:
std::cout << 5 << '\n'; // и это
}}
```

Выведет все значения больше 1: 2 3 4 5

2)

```
#include <iostream>
const int size = 5;
void Func(int *ptr, int size) {

for (int i = 0; i < size; ++i)
std::cout << ptr[i] << '\n';
*ptr = 5;
std::cout << '\n';
for (int i = 0; i < size; ++i)
std::cout << *(ptr++) << '\n';
*ptr = 55;
std::cout << '\n';
}
int main() {
int array[size]{1,3,5,7,9};
Func(array, size);
for (int i = 0; i < size; ++i)
std::cout << array[i] << '\n';
}
```

Выведет все значения массива: **`1,3,5,7,9`**

далее изменит первый элемент массива: **`5,3,5,7,9`**

в заключении выведет изменённый массив: **`5,3,5,7,9`**

Найдите ошибки в программах:

1)

```
#include <iostream>
int main() {
short array[5]{1,3,5,7,9};
short *ptr = array;
*ptr = 111;
//подсчитать кол-во элементов массива sizeof(array)/sizeof(array[0])
//for (int i = 0; i < sizeof(array); ++i)
//Правильный вариант
for (int i = 0; i < sizeof(array)/ sizeof(array[0]); ++i)
std::cout << array[i] << '\n';
}
2)
#include <iostream>
const int size = 5;
void Func(int *ptr, int size) {
for (int i = 0; i < size; ++i)
// std::cout << *ptr[i] << '\n';
//убрать символ указателя
std::cout << ptr[i] << '\n';

}
int main() {
int array[size]{1,3,5,7,9};

//Func(&array, size);
```

```
//усбрать символ ссылки

Func(array, size);

}
```

3)

```
#include <iostream>
int main() {
short value;
short *p;
p = value;    //вызовет ошибку
*p = value;   // правильная запись присвоения указателю переменной
*p = &value;  //вызовет ошибку
*p = *&value; //вызовет ошибку
}
```

4)

```
#include <iostream>
int main() {
short value, value1(3);
short &ref;    // Ошибка компилятора. некорректная ссылка. Ссылка должна ссылаться на что-нибудь
const short &ref1 = value;
const short &ref2 = 78;
ref1 = 3; // Ошибка: 3 - это r-value
*&value = 4;
const *short const p3; //Ошибка инициализации: тип данных не может быть укзателем
}
```