Урок 6 - контрольные вопросы

Специальные функции члены класса

- Конструктор
- Деструктор
- Конструктор копирования
- Перемещающий конструктор
- Оператор присваивания
- Оператор присваивания(перемещающий)

Перегрузка операторов

Нельзя перегружать(получим ошибку при компилляции):
- оператор разрешения области видимости `::`
- оператор размера объекта `sizeof`
- оператор выбора члена объекта `.`
- тернарный `?:`
- `.*`

Не рекомендуется перегружать операторы (классы, у которых перегружены данные операторы опасны):

- Оператор взятия адреса `&`
- Запятая `,`
- И `&&`
- Или `||`

Остальные операторы могут быть перегружены, но стоит учитывать возможность получить ошибку многократного определения.

Например. Оператор сложения уже реализован для целочисленных аргументов. Поэтому его перегрузка `operator+(int a, int b)` равноценная многократному определению.

Помимо этого при перегрузке число аргументов, принимаемых оператором, должно остаться неизменным. Изменить приоритет выполнения оператора тоже не является возможным.

Неявные преобразования типов

Например, мы перегружаем оператор `+` для нашего класса. И пусть второй класс имеет преобразование к объекту типа первого класса.

Тогда оператор `+` автоматически становится перегруженным для объектов второго класса(вследствие неявного преобразования типов).

Это может приводить к ошибкам в логике работы программы.

Классификация выражений

Выражения в C++ могут быть классифицированы как `rvalue`, `lvalue`,`xvalue`,`prvalue`,`glvalue`.

В старых версиях языка(младше одиннадцатой) существовало разделение только на `rvalue` и `lvalue`.

Свойством `lvalue` обладали выражения, адрес которых мог быть использован после выполнения выражения. Остальные выражения классифицировались как `rvalue`.

Например:
```
(a+b);
```

> Имеем дело с rvalue так как временный объект содержащий результат выражения будет удалён при выходе из области видимости выражения.

```
alpha = a+b;
```
> Это выражение уже lvalue

В новых версиях языка имеем больше свойств выражений:

- **lvalue** - нет изменений по сравнению со старыми версиями языка
- **xvalue** - результат выражения находится в конце своего жизненного цикла, но еще не удалён.
- **rvalue** - теперб состоит из **xvalue** и **prvalue**
- **prvalue** - тот **rvalue**, о котором говорилось ранее
- **glvalue** - состоит из **lvalue** и **xvalue**

rvalue ссылки

На объект стоит ссылаться при помощи rvalue ссылки в том случае, если его время жизни ограничивается областью видимости данного выражения.

Например:
```
s1 = s2+s3;
```
> имеем дело с prvalue выражением. Временный объект s2+s3 будет удалён после выполения данного выражения

В данном примере напрашивается использование сементики перемещения, котороя будет рассмотрена в следующем вопросе.

Тут отмечу, что реализовать семантику перемещения при помощи `lvalue` ссылки невозможно. Так как функция вида `func(s& ...)` не может быть выполнена с аргументом `s2+s3`.

Семантика перемещения

В некоторых ситуациях полезно перемещать данные одного объекта к другому объекту. При этом использовать минимум операций копирования.

В примере из предыдущего вопроса полностью копировать данные временного объекта `s2+s3` неуместно. Ведь временный объект не может быть использован далее(в ходе работы программы).

Поэтому наилучший способ присвоения результата выражения объекту `s1` - переместить данные временного объекта в объект `s1`.

Таким образом мы избавляемся от лишних затрат времени при копировании.

std::move

`std::move(obj)` используется для индикации того, что объект `obj` может быть перемщен куда либо. Другими словами `std::move` превращает lvalue ссылку в rvalue ссылку.

Пусть, например, у класса `A` есть перемещающий оператор присваивания и копирующий оператор присваивания.

```
A alpha;
A beta;
...
```

alpha = beta;
```
> Тут вызывается оператор присваивания копирования, принимающий lvalue ссылку

```
A alpha;
A beta;
...
alpha = std::move(beta);
```
> Здесь будет вызван оператор присваивания перемещения

То есть std::move используется тогда, когда мы явно хотим указать об использовании семантики перемещения.

Если выражение обладает свойством `rvalue`, то объект и так будет преобразован к rvalue ссылке. Поэтому использовать std::move необязательно.

# Специальные функции члены

Реализация специальных функций членов требуется при работе с динамически выделенными данными, или объектами тип которых содержит динамически выделенные данные.

# default и delete

`default` используется для того, чтобы указать компилятору сгенерировать данную специальную функцию член самостоятельно.
`delete` используется для того, чтобы указать отсутствие данной специальной функции члена в классе.
```