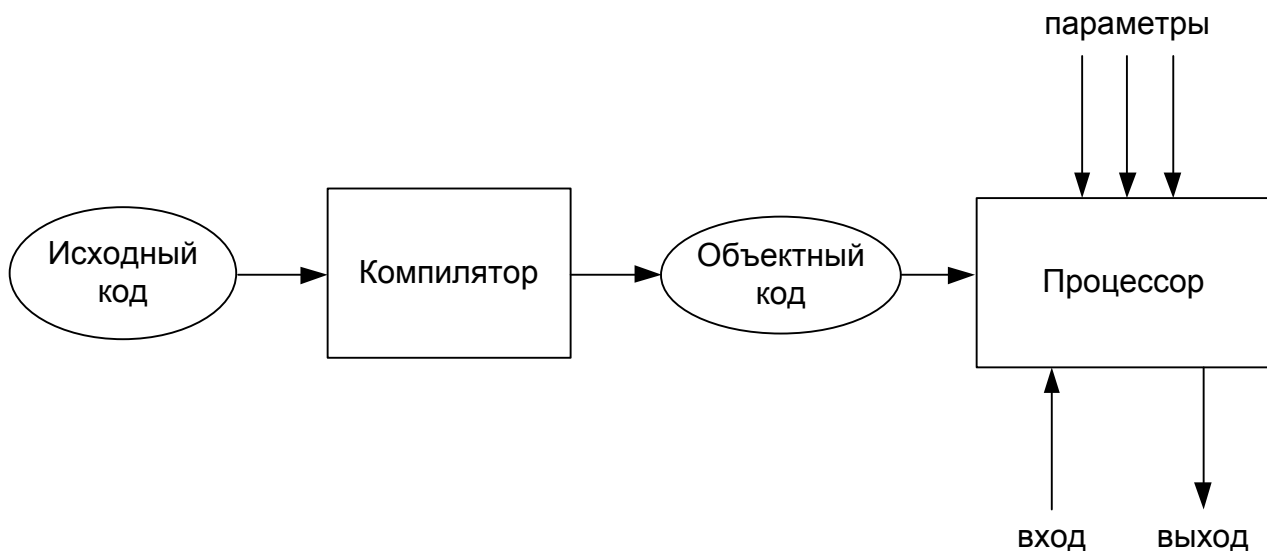


Генерация кода. Интерпретаторы

Подходы к разработке трансляторов:

- часть операционной системы;
- для аппаратной платформы (ассемблер);
- реализации для одной программной платформы;
- реализация для одной программной платформы, но для разных процессоров;
- интерпретаторы;
- несколько реализаций для разных платформ;
- кроссплатформенные реализации (Java);
- компиляторы-интерпретаторы (компиляция + интерпретация);
- разработка стандарта и стандартизация (Java, C++, C#)

1. Схема работы «чистого» компилятора:



Для каждой целевой машины и каждой операционной системы или семейства операционных систем, работающих на целевой машине, требуется написание своего компилятора.

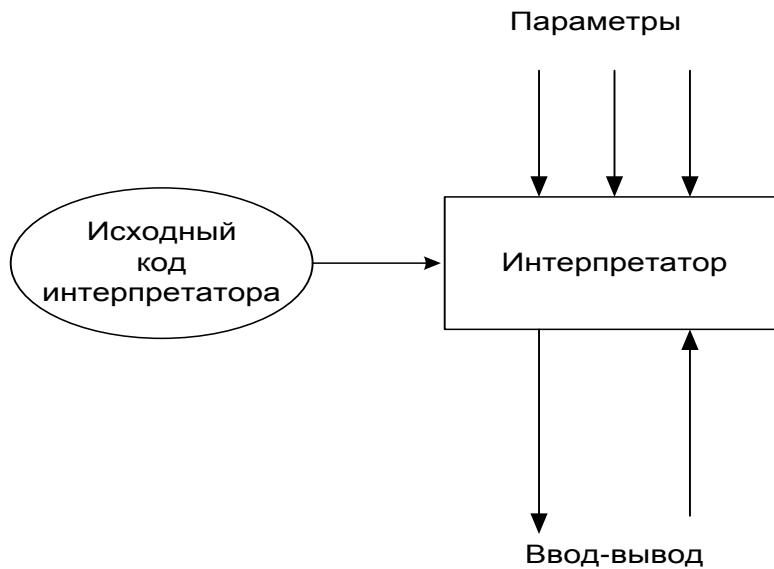
2. Компиляторы могут транслировать исходный код в язык ассемблера для некоторой аппаратной платформы

Цель: упростить генерацию кода.

3. Интерпретаторы

Интерпретатор – программа или устройство, осуществляющее пооператорную трансляцию и выполнение исходной программы.

Схема работы интерпретатора:



Пример интерпретации:

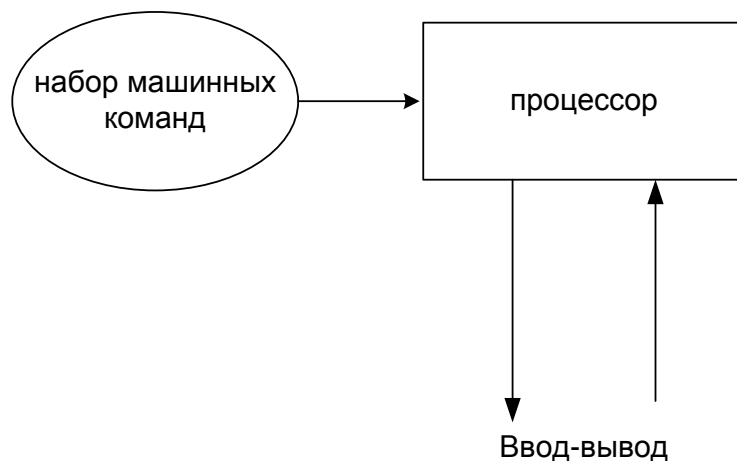
$x := x + 1$

Действия интерпретатора	
взять значение x ;	
вычислить новое значение: значение $x + 1$;	
заменить результат в x	

ТИ	0
имя	x
тип	int
иниц.	0x00000000

ТИ	1
имя	L001
тип	int
иниц.	0x00000001

4. Процессор – это интерпретатор машинных команд

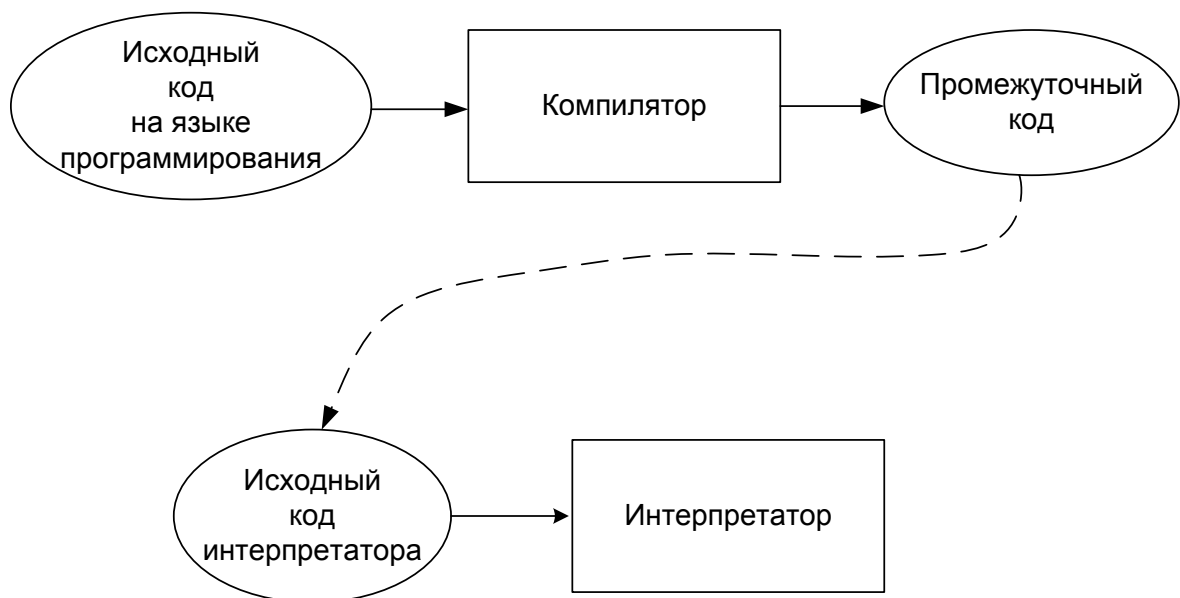


5. Интерпретатор – программная реализация процессора, поэтому часто интерпретаторы так и называют – процессоры.

Структура интерпретатора включает в себя фазы лексического и синтаксического анализа. Интерпретатор может транслировать исходную программу во внутреннюю форму, а затем выполнять программу в этой форме.

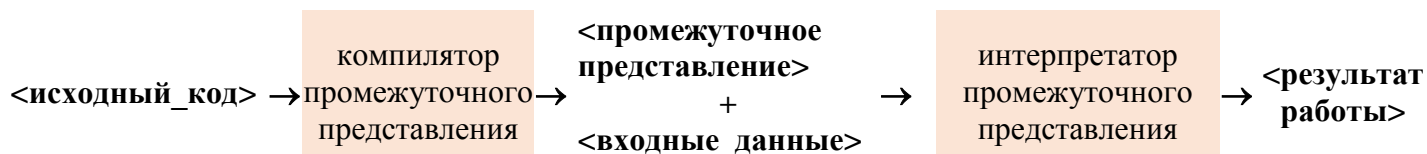
Схемы чистой компиляции и чистой интерпретации являются крайними вариантами. Большинство практических решений представляют собой их сочетание.

6. Компиляторы-интерпретаторы: сначала генерируется промежуточный код, затем он интерпретируется (для ускорения работы).



7. Кросс-компиляторы позволяют на одной машине и в среде одной ОС генерировать код, предназначенный для выполнения на другой целевой машине и/или в среде другой ОС.

Смешанная стратегия предполагает, что компилятор создает код на промежуточном языке, который является входным для некоторой виртуальной машины. Такой подход объединяет преимущества компиляции и интерпретации.

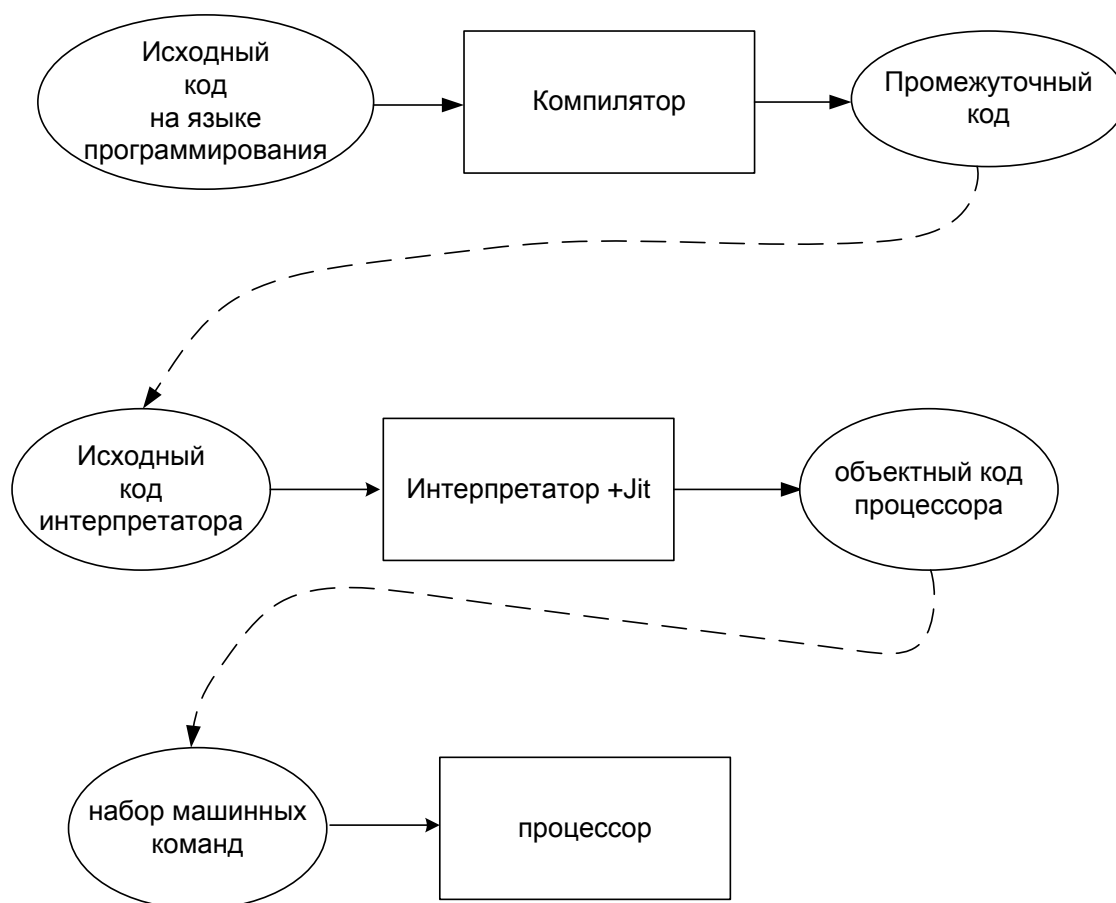


Генерация байт-кода

Байт-код – это подход, при котором программа преобразуется в промежуточный двоичный вид, интерпретируемый некой «виртуальной машиной» во время исполнения.

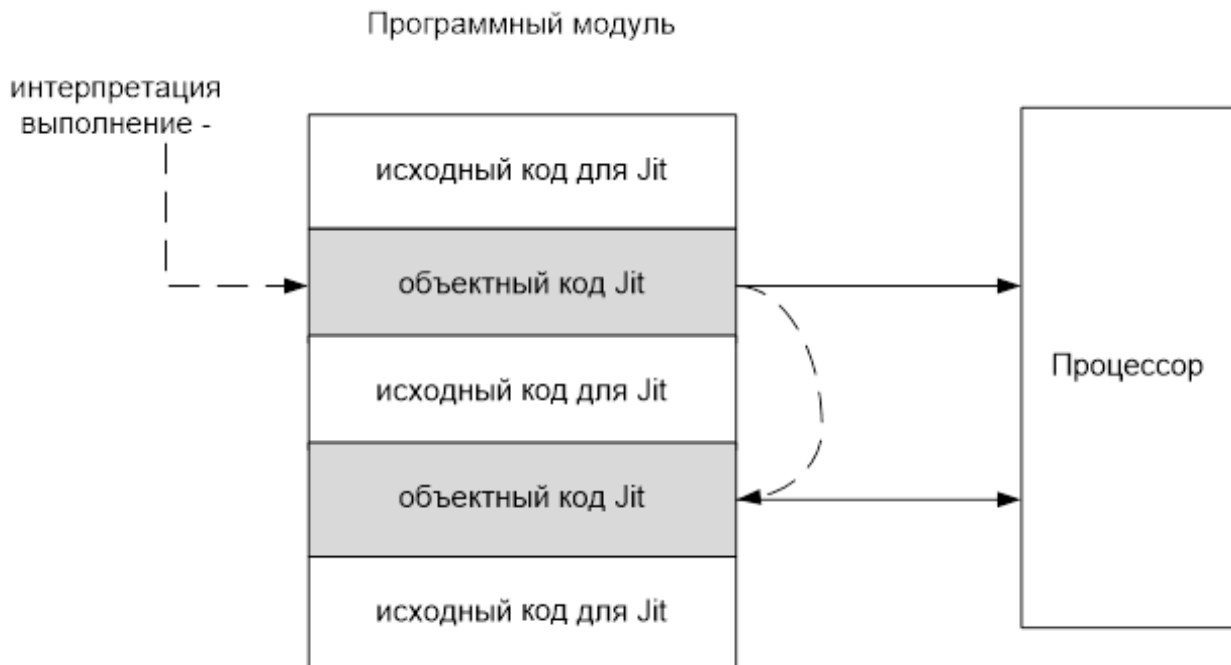
8. Компиляторы-интерпретаторы

Jit-трансляторы: сначала генерируется промежуточный код, затем он компилируется в объектный код аппаратной платформы. Jit-трансляторы могут осуществлять частичную трансляцию по мере необходимости.

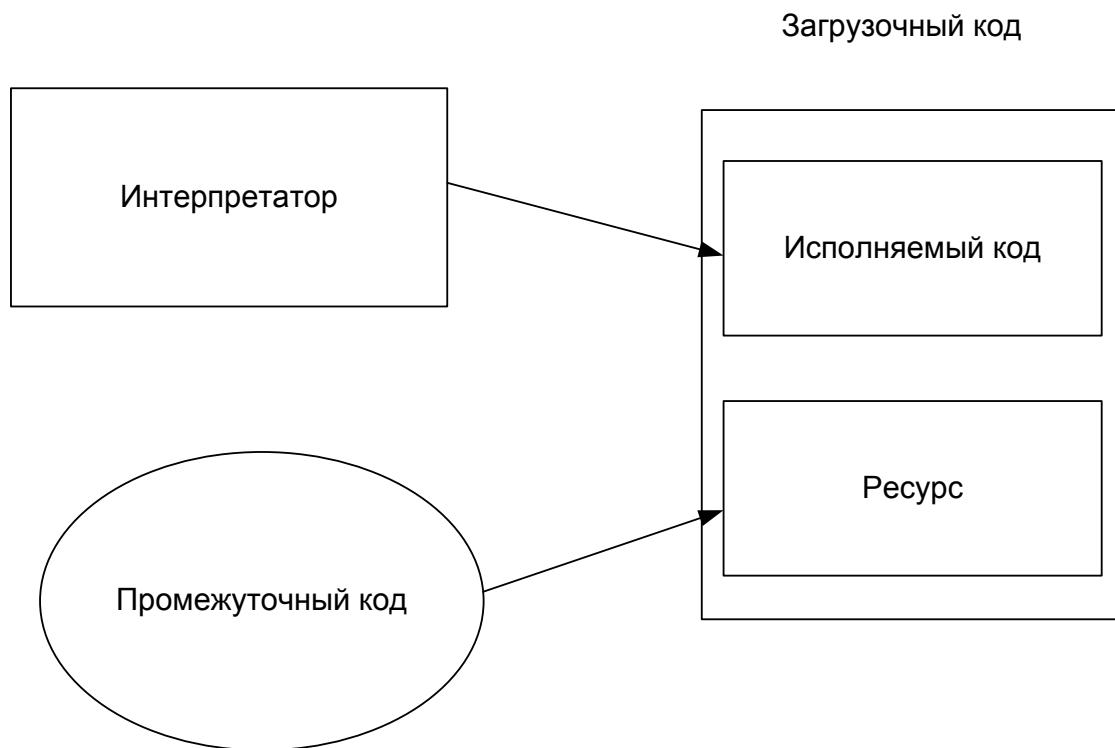


9. Частичная компиляция

После внесения изменений компилируются только те части программы, которые были модифицированы после предыдущей компиляции.



10. Объединение объектного кода с интерпретатором



Пример подхода к реализации генерации промежуточного кода для последующей интерпретации

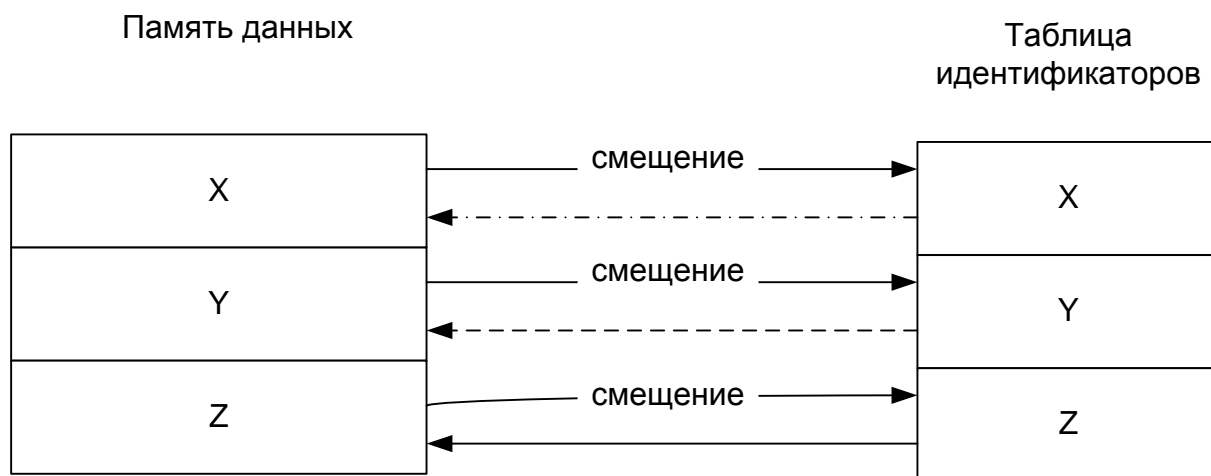
11. Последовательность разработки:

- 1) построить план (модель) памяти;
- 2) определить перечень инструкций промежуточного кода;
- 3) разработать генератор кода;
- 4) разработать интерпретатор.

12. Построение плана памяти: плоская память.



13. Построение плана памяти: память данных (принцип лезвия Оккама) «Не следует множить сущее без необходимости»



где смещение определяется исходя из ширины типов данных.

14. Построение плана памяти: память данных – это ячейки памяти

тип	данные
тип	данные
тип	данные

Строится развернутый план памяти с *мета-данными*: ячейки памяти хранят тип переменной или литерала и непосредственно сами данные.

15. Построение плана памяти для целочисленных данных

Пример. Память данных – ячейки памяти для целочисленных данных (пустая ячейка и ячейка с литералом)

0x01	0x00000000
------	------------

0x01	0x00000016
------	------------

16. Построение плана памяти для строковых данных

Пример. Память данных – ячейки памяти для строковых данных (пустая ячейка и ячейка со строковым литералом указанной длины)

0x02	0x00	0x00 0x00 0x00 ...
------	------	--------------------

0x02	0x03	0x61 0x62 0x63 0x00 ...
------	------	-------------------------

17. Сериализация памяти данных

0x01	0x00000000	0x01	0x00000016	0x02	0x00	0x02	0x03	0x61 0x62 0x63
------	------------	------	------------	------	------	------	------	----------------

18. Реализация в C++ ячейки памяти

```
struct TYPEINT    // целочисленные данные
{
    int data;
};
```

```
struct TYPESTR    // строковые данные
{
    unsigned char len;
    char data[255];
};
```

```
struct CELL    // ячейка памяти
{
    enum CELLTYPE {INT=0x01, STR=0x02};

    CELLTYPE celltype;
    void* data;
    CELL (CELLTYPE celltype)    // пустая ячейка
    {
        this->celltype = celltype;
        switch (celltype)
        {
            case CELLTYPE::INT: this->data = new TYPEINT; ((TYPEINT*) this->data)->data = 0; break;
            case CELLTYPE::STR: this->data = new TYPESTR; ((TYPESTR*)this->data )->len = 0x00; break;
        }
    }

    CELL (int data)    // для литерала
    {
        this->celltype = CELLTYPE::INT;
        this->data = new TYPEINT; ((TYPEINT*) this->data)->data = data;
    }

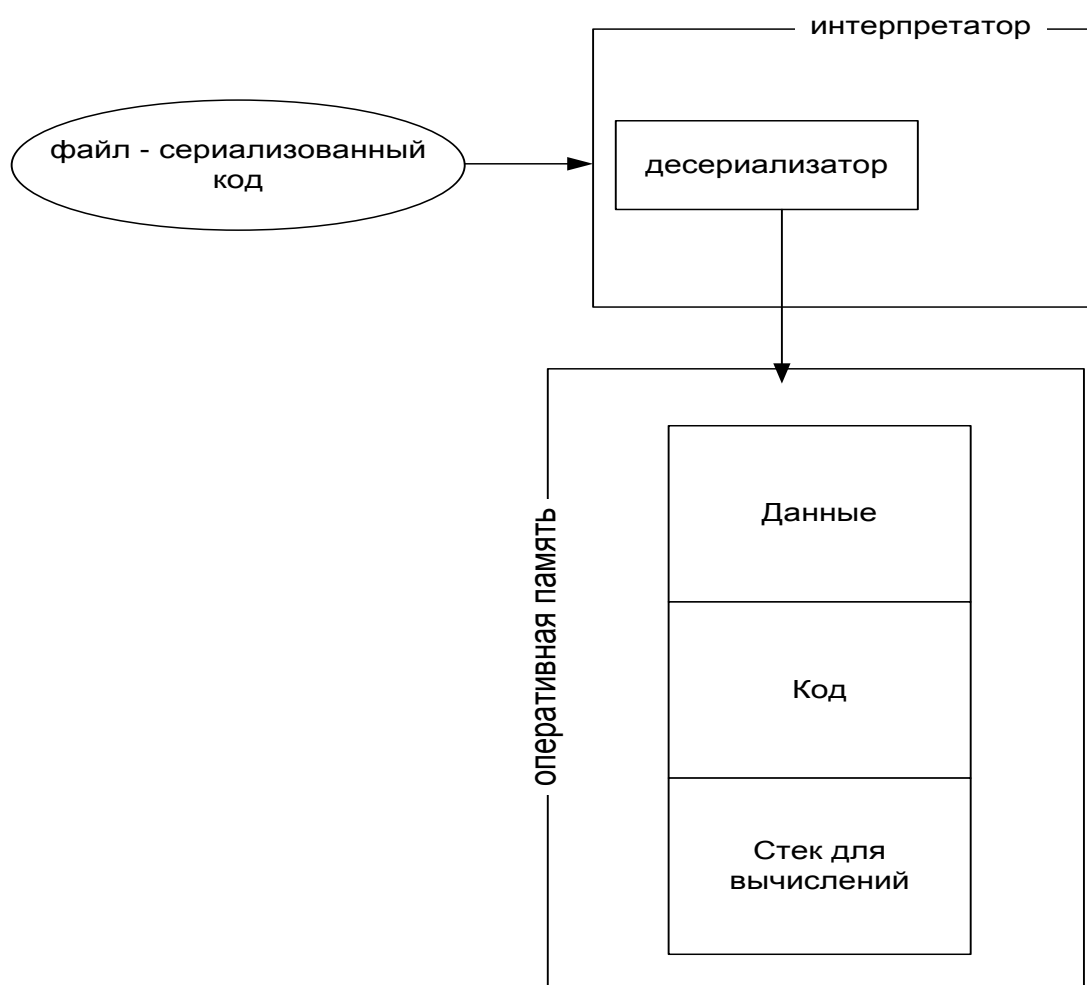
    CELL (char* data)    // для литерала
    {
        this->celltype = CELLTYPE::STR;
        int l = strlen(data);
        this->data = new TYPEINT; ((TYPESTR*) this->data)->len = l = (l < 256?l: 255);
        strcpy_s( (char*)this->data, l, data);
    }
}
```

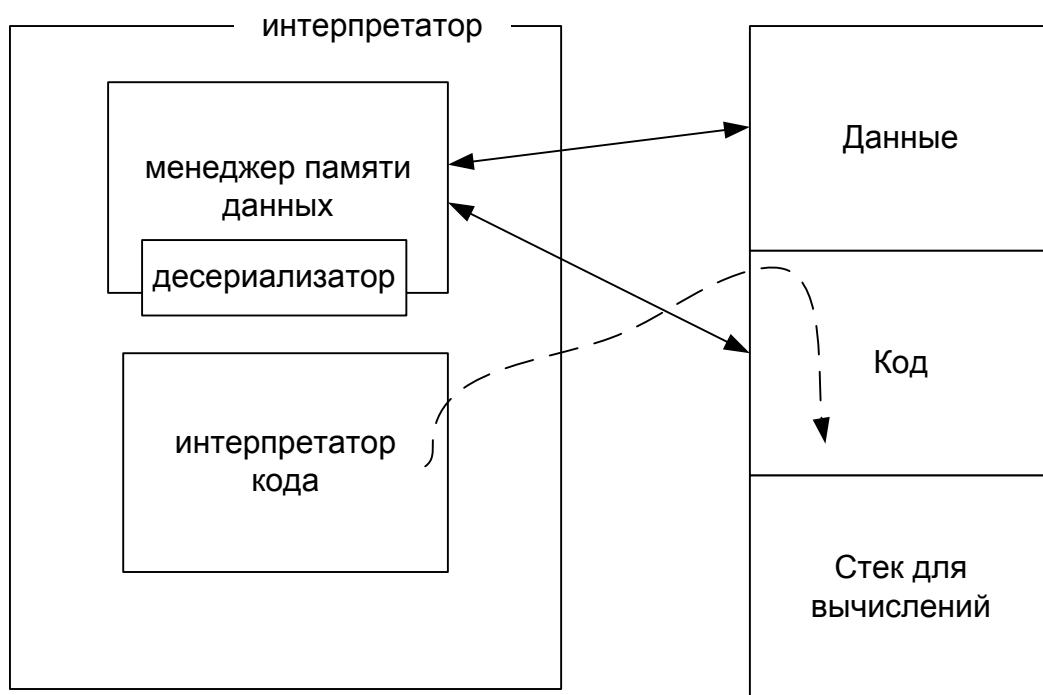
```

char* serialize()           // сериализация ячейки памяти
{
    char* rc = nullptr;
    switch(this->celltype)
    {
        case CELLTYPE::INT: rc = new char[5];
                             rc[0] = (char)CELLTYPE::INT;
                             memcpy_s(rc+1,4, (void*)((TYPEINT*)this->data)->data,4);
                             break;
        case CELLTYPE::STR: rc = new char[257];
                             rc[0] = (char)CELLTYPE::STR;
                             rc[1] = (char)((TYPESTR*)this->data)->len;
                             memcpy_s(rc+2, (int)rc[1],(void*)((TYPESTR*)this->data)->data,255);
                             break;
    }
    return rc;
}
};

```

19. Менеджер памяти данных – программный код, обеспечивающий доступ к памяти.





20. Доступ к данным через смещения: при размещении десериализованных данных.

Менеджер памяти данных запоминает адрес первой ячейки, адрес любой ячейки может быть определен (вычислен) через смещение.

21. Инструкции промежуточного кода

Данные
инструкция 1
инструкция 2
инструкция 3
.....
инструкция N
Стек для вычислений

22. Инструкция (тетрада)

код инструкции	операнд 1	операнд 2	операнд 3
-------------------	-----------	-----------	-----------

23. Пример инструкции целочисленного сложения.

Код операции 0x01

Операнды: задействован только первый операнд.

Действие: инструкция извлекает из *стека вычислений* целочисленное число и складывает его с целым числом, *смещение* которого указывается первым операндом.

0x01	0x00000100	0x00000000	0x00000000
------	------------	------------	------------

Результат заносится в стек.

24. Пример инструкции пересылки строковых данных сложения.

Код операции 0x02.

Операнды: задействованы два операнда; первый – строка приемник, второй – строка источник.

Действие: инструкция побайтно пересылает данные строки, *смещение* которой указывается вторым операндам, в строку, *смещение* которой указывается первым операндом.

0x02	0x00000100	0x00003220	0x00000000
------	------------	------------	------------

25. Пример инструкции безусловного перехода

Код операции 0x03.

Операнды: задействован один операнд.

Действие: управление передается инструкции, *номер* которой указан в операнде.

0x03	0x00000005	0x00000000	0x00000000
------	------------	------------	------------

26. Пример инструкции условного перехода.

Код операции 0x04.

Операнды: задействованы два операнда; первый – операнд указывает номер инструкции, на которую следует передать управление, второй – смещение.

Действие: управление передается инструкции с указанным *номером* в случае, если значение, указанное *смещением* во втором операнде, указывает на 4 байта, содержащие только нулевые биты, в противном случае, если хотя бы один бит равен единице, то осуществляется переход на следующую по порядку инструкцию.

0x04	0x00000005	0x00000102	0x00000000
------	------------	------------	------------

27. Стек для вычислений.

Стек состоит из ячеек для хранения *смещений*.

Поместить в стек данные – означает поместить в стек *смещение* данных.

Извлечь данные из стека – означает извлечь из стека данные по указанному смещению.

Доступ к самим данным осуществляется через смещения.

Стек для вычислений состоит из однородных ячеек, длиной 4 байта.