Лекция 8.

Базовые элементы языков программирования

Базовые элементы ЯП

- Переменные
- Структуры управления
  - Последовательность
  - Выбор
  - Цикл
- Подпрограммы

Переменные

В императивных языках **переменная** – это поименованная область памяти, адрес которой можно использовать для осуществления доступа к данным.

В функциональных языках переменной может являться функция.

Переменные

Переменная – некоторый объект, содержащий данные, которые могут изменяться.
Константа – переменная, которая не меняется.

На переменные действует оператор присваивания:

target := *source*

Переменные

По времени создания: **статические** и **динамические**.

- Статические создаются в момент запуска программы
- Динамические - в процессе ее выполнения (если заранее неизвестно количество элементов).

Динамические структуры: стек, очередь и т.п.

Переменные

По наличию внутренней структуры бывают **простыми** и **составными.**

```
var r: real;
```
// Простая переменная в Pascal

Переменные

По наличию внутренней структуры бывают **простыми** и **составными.**

```
var r: real;
```
// Простая переменная в Pascal

Внутренности числа с плавающей запятой.

Три части числа с плавающей запятой.

Типичные составные переменные

- **Массив** (элементы однотипные)

```
var a: array [1..10] of integer;
```

- **Запись** (элементы могут быть разных типов)

```
Type TDate  = Record
      Day: 1..31;
      Month: 1..12;
      Year: 1..2100;
end;
var Date: TDate;
```

- **Ассоциативный массив** (словарь, хэш-таблица). Содержит пары вида "ключ-значение"

```
d = {'a': 10, 'b': 20}  # Python
$d = array('a'=>10, 'b'=>20); //PHP
```

Использование переменных в программе

Ясность программы важнее ее краткости.

Использование переменных в программе

Ясность программы важнее ее краткости.

- Явное объявление переменных.
- Правильная их инициализация.
- Минимизация области видимости переменных.
- Тщательный выбор имен переменных.

Философия программирования "The Zen of Python"

- Красивое лучше, чем уродливое.
- Явное лучше, чем неявное.
- Простое лучше, чем сложное.
- Сложное лучше, чем запутанное.
- Плоское лучше, чем вложенное.
- Разреженное лучше, чем плотное.
- Читаемость имеет значение.
- Особые случаи не настолько особые, чтобы нарушать правила.
- При этом практичность важнее безупречности.
- Ошибки никогда не должны замалчиваться.
- Если не замалчиваются явно.
- Встретив двусмысленность, отбрось искушение угадать.
- Должен существовать один — и, желательно, *только* один — очевидный способ сделать это.
- Если реализацию сложно объяснить — идея плоха.
- Если реализацию легко объяснить — идея, *возможно*, хороша.

Объявление переменных

Переменные могут объявляться **явно** или **неявно**.

При объявлении переменной компилятор выделяет место в памяти, необходимое для размещения переменной данного типа.

```
int a, b, c;
float a1,a2;
```

Неявные объявления - источник ошибок.

Объявление переменных

Рекомендации:

- **Отключите, если это возможно, неявные объявления.** Например, в Visual Basic есть директива *Option Explicit On*.
- **Объявляйте все переменные**, даже если язык программирования этого не требует.
- **Выберите стандарт наименования** и придерживайтесь его.

Объявление переменных

Пример: *Нужно определить переменную, содержащую номер счета (account number).*

`AccountNumber`, `AccNum`, `AccNom`, `AcntNumber`, `AcntNomer`, …

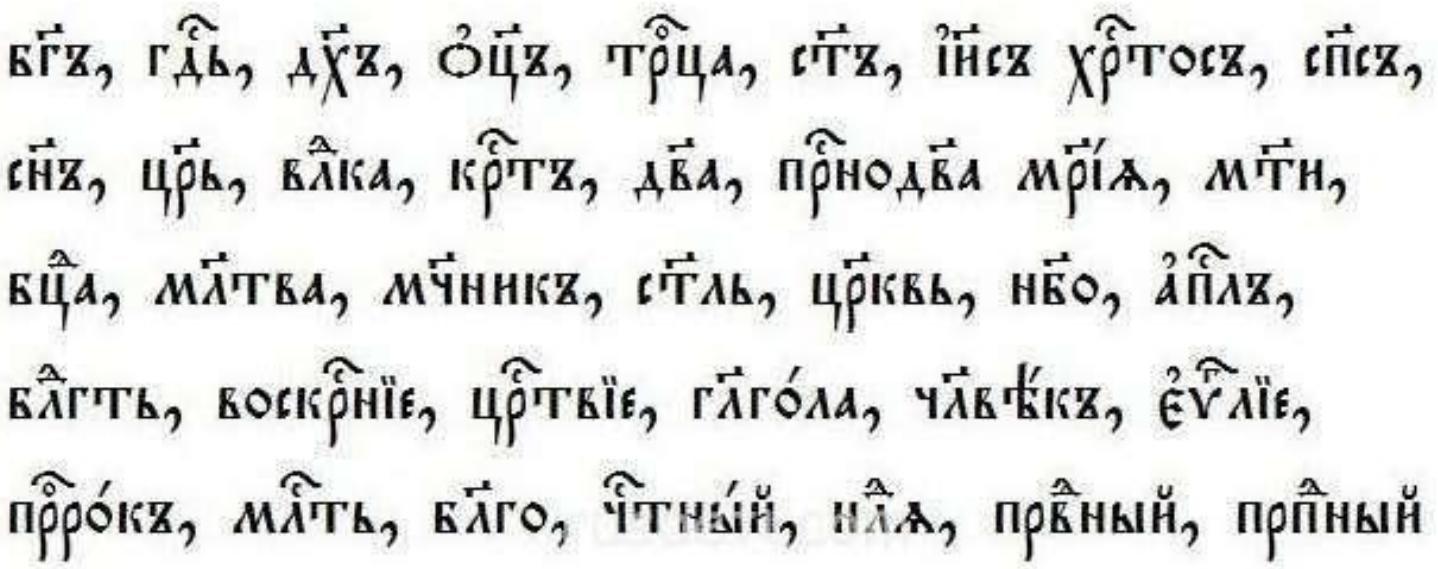
Нужно ли сокращать и как правильно это сделать?

Объявление переменных

Пример: *Нужно определить переменную, содержащую номер счета (account number).*

`AccountNumber`, `AccNum`, `AccNom`, `AcntNumber`, `AcntNomer`, …

Нужно ли сокращать и как правильно это сделать?



Инициализация переменных

Инициализация = начальное присвоение значений.

Неверная инициализация – источник ошибок.

Инициализация переменных

Рекомендации:

- **Инициализируйте каждую переменную при ее объявлении.**

```
int a=5, b=10;
```

Если это невозможно, то …

- **Инициализируйте каждую переменную там, где она используется в первый раз.**
- **Объявляйте переменные как можно позже.** В идеальном случае сразу объявляйте и определяйте каждую переменную непосредственно перед первым обращением к ней.

Инициализация переменных

Принцип близости:

Связанные действия должны быть сгруппированы вместе.

- Определение переменной – около ее использования.
- Комментарии – около описываемого кода.
- Код настройки цикла – около самого цикла.
- И т.п.

Область видимости переменной

Область видимости – фрагмент программы, в котором переменная известна и может быть использована.

Область видимости переменной

Область видимости – фрагмент программы, в котором переменная известна и может быть использована.

Какая область видимости для переменной лучше:

- большая (в пределе – вся программа или проект) или
- маленькая (например, только цикл)?

Область видимости переменной

Область видимости – фрагмент программы, в котором переменная известна и может быть использована.

Какая область видимости для переменной лучше:

- большая (в пределе – вся программа или проект) или
- маленькая (например, только цикл)?

Большая область видимости переменной – удобней и быстрее писать программу, но тяжелее понимать, отлаживать и изменять.

Область видимости переменной

Область видимости – фрагмент программы, в котором переменная известна и может быть использована.

Какая область видимости для переменной лучше:

- большая (в пределе – вся программа или проект) или
- маленькая (например, только цикл)?

Большая область видимости переменной – удобней и быстрее писать программу, но тяжелее понимать, отлаживать и изменять.

На что мы ориентируемся: на написание программы (одноразовый код – написал и забыл) или на ее чтение?

Область видимости переменной

Минимизация области видимости переменных:

- Не нужно делать переменные глобальными, если в этом нет необходимости.
- Если переменные можно сделать локальными для функции, не объявляйте их на уровне файла.
- Если переменные можно сделать локальными для цикла, не объявляйте их на уровне функции.

Время связывания переменной

Время связывания (time bounding) – момент, когда переменная и ее значение связываются вместе.

Время связывания переменной

Пример: *задание значения переменной* titleBar.color, *в которой хранится цвет заголовка*.

Время связывания переменной

Пример: *задание значения переменной* titleBar.color, *в которой хранится цвет заголовка*.

- **Связывание во время написания кода.**

```
titleBar.color = 0xFF;
```

Время связывания переменной

Пример: *задание значения переменной* titleBar.color, *в которой хранится цвет заголовка*.

- **Связывание во время написания кода.**

```
titleBar.color = 0xFF;
```

- **Связывание во время компиляции.**

```
private static final int COLOR_BLUE = 0xFF;
private static final int TITLE_BAR_COLOR = COLOR_BLUE;
...
titleBar.color = TITLE_BAR_COLOR;
```

Время связывания переменной

Пример: *задание значения переменной* titleBar.color, *в которой хранится цвет заголовка.*

- **Связывание во время написания кода.**

```
titleBar.color = 0xFF;
```

- **Связывание во время компиляции.**

```
private static final int COLOR_BLUE = 0xFF;
private static final int TITLE_BAR_COLOR = COLOR_BLUE;
...
titleBar.color = TITLE_BAR_COLOR;
```

- **Связывание в период исполнения.**

```
titleBar.color = ReadTitleBarColor();
```

Метод может вызываться:

- При загрузке программы (чтение значения из реестра).
- При создании объекта.
- По требованию (при каждой перерисовке).

Время связывания переменной

Пример: *задание значения переменной* titleBar.color, *в которой хранится цвет заголовка.*

- **Связывание во время написания кода.**

```
titleBar.color = 0xFF;
```

- **Связывание во время компиляции.**

```
private static final int COLOR_BLUE = 0xFF;
private static final int TITLE_BAR_COLOR = COLOR_BLUE;
...
titleBar.color = TITLE_BAR_COLOR;
```

- **Связывание в период исполнения.**

```
titleBar.color = ReadTitleBarColor();
```

Метод может вызываться:

- При загрузке программы (чтение значения из реестра).
- При создании объекта.
- По требованию (при каждой перерисовке).

Чем раньше переменная свяжется со своим значением, тем проще код, но ниже гибкость.

Рекомендации по использованию переменных

- **Используйте каждую переменную только с одной целью.**
- **Избегайте переменных, имеющих скрытый смысл.**

  Так плохо:

  `pageCount>=0` - количество отпечатанных страниц,

  `pageCount=-1` - ошибка при печати
- **Используйте все объявленные переменные.**

Выбор имен переменных

Как вы яхту назовете, так она и поплывет

Выбор имен переменных

Как вы яхту назовете, так она и поплывет

Переменная и ее имя - одна сущность

Схемы именования переменных

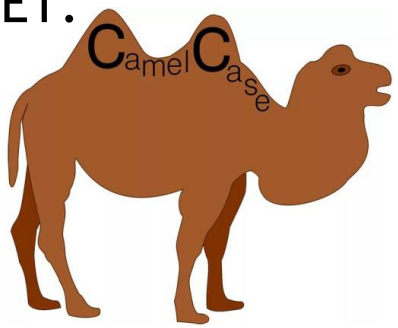
Называть переменные можно по-разному, есть несколько общепринятых соглашений об именовании.

Они помогают:

- визуально выделить составные части имени;
- понять суть переменной, ее назначение и тип (именованная константа; элементарная переменная; тип, определенный пользователем, или класс).

Схемы именования переменных

- **CapitalizedWords** (CamelCase, PascalCase). Применяется для имен классов Java, в методах Windows API и .NET.



- **mixedCase**. Применяется для членов классов в Java.

- **low_case_under_score**. Принят в PHP.
- **UPPERCASE_WITH_UNDERSCORES**. Для наименования констант.
- **_name**. Подчеркивание перед именем указывает на то, что переменная объявлена как private или protected и недоступна вне класса.

Схемы именования переменных

Венгерская нотация – правило, требующее включать в имя переменной или функции данные об их типе.

Схемы именования переменных

Венгерская нотация – правило, требующее включать в имя переменной или функции данные об их типе.

Префикс	Сокращение от	Смысл	Пример
s	string	строка	`sClientName`
sz	zero-terminated string	строка, ограниченная нулевым символом	`szClientName`
n, i	int	целочисленная переменная	`nSize, iSize`
l	long	длинное целое	`lAmount`
b	boolean	булева переменная	`bIsEmpty`
a	array	массив	`aDimensions`
t, dt	time, datetime	время, дата и время	`tDelivery`, `dtDelivery`
p	pointer	указатель	`pBox`
lp	long pointer	двойной (дальний) указатель	`lpBox`
r	reference	ссылка	`rBoxes`
h	handle	дескриптор	`hWindow`
m_	member	переменная-член	`m_sAddress`
g_	global	глобальная переменная	`g_nSpeed`
C	class	класс	`CString`
T	type	тип	`TObject`
I	interface	интерфейс	`IDispatch`
v	void	отсутствие типа	`vReserved`

Схемы именования переменных

Нужно придерживаться определенного стиля именования (в соответствии с традициями языка).

Главное - единообразие!

Схемы именования переменных

Нужно придерживаться определенного стиля именования (в соответствии с традициями языка).

Главное – единообразие!

```
class badly_named : public MyBaseClass
{
public:
    void doTheFirstThing();
    void DoThe2ndThing();
    void do_the_third_thing();
};
```

Выбор имен переменных

- **Имена должны быть максимально конкретны.** Для важных переменных следует избегать имен x, j, temp, … . Короткие имена допустимы в качестве счетчиков в циклах и т.п.

Выбор имен переменных

- **Имена должны быть максимально конкретны.** Для важных переменных следует избегать имен x, j, temp, … . Короткие имена допустимы в качестве счетчиков в циклах и т.п.
- **Следует отказаться от сокращения имен.** Нужно выбирать имена так, чтобы они облегчали чтение кода, даже за счет удобства его написания.

Выбор имен переменных

- **Имена должны быть максимально конкретны.** Для важных переменных следует избегать имен x, j, temp, … . Короткие имена допустимы в качестве счетчиков в циклах и т.п.
- **Следует отказаться от сокращения имен.** Нужно выбирать имена так, чтобы они облегчали чтение кода, даже за счет удобства его написания.

Пример: *В переменной хранится текущая дата. Как назвать переменную?*

CD, CurDate, CurrentDate.

Выбор имен переменных

- **Имена должны быть максимально конкретны.** Для важных переменных следует избегать имен x, j, temp, … . Короткие имена допустимы в качестве счетчиков в циклах и т.п.
- **Следует отказаться от сокращения имен.** Нужно выбирать имена так, чтобы они облегчали чтение кода, даже за счет удобства его написания.

Пример: *В переменной хранится текущая дата. Как назвать переменную?*

CD, CurDate, CurrentDate.

Выбор имен переменных

Хорошее имя чаще всего описывает проблему, а не ее решение (отвечает на вопрос ***что?*** а не ***как?***)

Выбор имен переменных

Хорошее имя чаще всего описывает проблему, а не ее решение (отвечает на вопрос ***что?*** а не ***как?***)

Запись данных о сотруднике: *inputRecord* или *employeeData*

Статус принтера: *bitFlag* или *printerReady*

Выбор имен переменных

Хорошее имя чаще всего описывает проблему, а не ее решение (отвечает на вопрос ***что?*** а не ***как?***)

Запись данных о сотруднике: *inputRecord* или *employeeData*

Статус принтера: *bitFlag* или *printerReady*

Выбор имен переменных

Следует избегать:

- **Аббревиатур или неоднозначных имен.**
  data, value, temp

Выбор имен переменных

Следует избегать:

- **Аббревиатур или неоднозначных имен.**
  data, value, temp
- **Имен, имеющих похожие значения (синонимы).**
  number и nomer

Выбор имен переменных

Следует избегать:

- **Аббревиатур или неоднозначных имен.**
  data, value, temp
- **Имен, имеющих похожие значения (синонимы).**
  number и nomer
- **Переменных, имеющих разную суть, но похожие имена.**
  clientRecs, clientReps

Выбор имен переменных

Следует избегать:

- **Аббревиатур или неоднозначных имен.**
  data, value, temp
- **Имен, имеющих похожие значения (синонимы).**
  number и nomer
- **Переменных, имеющих разную суть, но похожие имена.**
  clientRecs, clientReps
- **Имен, содержащих цифры** (лучше массивы).

Выбор имен переменных

Следует избегать:

- **Аббревиатур или неоднозначных имен.**
  data, value, temp
- **Имен, имеющих похожие значения (синонимы).**
  number и nomer
- **Переменных, имеющих разную суть, но похожие имена.**
  clientRecs, clientReps
- **Имен, содержащих цифры** (лучше массивы).
- **Имен, отличающихся регистром символов.**

Выбор имен переменных

Следует избегать:

- **Аббревиатур или неоднозначных имен.**
  data, value, temp
- **Имен, имеющих похожие значения (синонимы).**
  number и nomer
- **Переменных, имеющих разную суть, но похожие имена.**
  clientRecs, clientReps
- **Имен, содержащих цифры** (лучше массивы).
- **Имен, отличающихся регистром символов.**
- **Имен, совпадающих со стандартными типами, переменными и методами.**
  integer, string

Выбор имен переменных

Следует избегать:

- **Аббревиатур или неоднозначных имен.**
  data, value, temp
- **Имен, имеющих похожие значения (синонимы).**
  number и nomer
- **Переменных, имеющих разную суть, но похожие имена.**
  clientRecs, clientReps
- **Имен, содержащих цифры** (лучше массивы).
- **Имен, отличающихся регистром символов.**
- **Имен, совпадающих со стандартными типами, переменными и методами.**
  integer, string
- **Имен, которые не связаны со смыслом переменных.**

Выбор имен переменных

Следует избегать:

- **Аббревиатур или неоднозначных имен.**
  data, value, temp
- **Имен, имеющих похожие значения (синонимы).**
  number и nomer
- **Переменных, имеющих разную суть, но похожие имена.**
  clientRecs, clientReps
- **Имен, содержащих цифры** (лучше массивы).
- **Имен, отличающихся регистром символов.**
- **Имен, совпадающих со стандартными типами, переменными и методами.**
  integer, string
- **Имен, которые не связаны со смыслом переменных.**

Никогда не использовать символы I, l, O как однобуквенные идентификаторы!

Выбор имен переменных

Хорошие программисты...

- Понимают важность выбора имен и занимаются этой проблемой
- Не забывают о выборе правильного имени для каждого создаваемого объекта
- Учитывают многие факторы: длину имени, понятность, контекст и т. д.
- Видят общую картину и выбирают имена в рамках проекта (или проектов)

Плохие программисты...

- Не заботятся о понятности своего кода
- Пишут *одноразовый* код, плохо продумывая его в погоне за скоростью
- Игнорируют естественную идиоматику языка
- Не стремятся к единообразию в именах
- Не заботятся об общей картине и не интересуются согласованностью своего кода с проектом в целом

Выбор имен переменных

Хорошо ли выбраны имена переменных?

a. `int apple_count`
b. `char foo`
c. `bool apple_count`
d. `char *string`
e. `int loop_counter`

Пример

Что делает этот код, какой алгоритм реализует?

```
void bsrt(int a[], int n)
{
    for (int i = 0; i < n1; i++)
        for (int j = n1; j > i; j)
            if (a[j1] > a[j])
            {
                int tmp = a[j1];
                a[j1] = a[j];
                a[j] = tmp;
            }
}
```

Пример

Что делает этот код, какой алгоритм реализует?

```
void bsrt(int a[], int n)
{
    for (int i = 0; i < n1; i++)
        for (int j = n1; j > i; j)
            if (a[j1] > a[j])
            {
                int tmp = a[j1];
                a[j1] = a[j];
                a[j] = tmp;
            }
}
```

```
void swap(int *first, int *second)
{
    int temp = *first;
    *first = *second;
    *second = temp;
}

void bubbleSort(int items[], int size)
{
    for (int pos1 = 0; pos1 < size1;
pos1++)
        for (int pos2 = size1; pos2 > pos1;
pos2)
            if (items[pos21] > items[pos2])
                swap(&items[pos21],
&items[pos2]);
}
```

Структуры управления

Структура управления – программная конструкция, воздействующая на поток управления.

Определяют последовательность действий при выполнении программы.

Структуры управления

Базовые структуры управления:

- **Последовательность** – операторы, перечисленные в определенном порядке. Ее выполнение состоит в выполнении каждого из этих операторов в том же порядке.
- **Цикл**, который содержит последовательность операторов, выполняемую многократно.
- **Выбор**, состоящий из условия и двух последовательностей операторов.

Структуры управления

Дополнительные структуры управления:

- **Оператор перехода (goto).** Любая программа, использующая его, имеет эквивалент, выраженный в терминах стандартных структур управления.
- **Обработка исключений**, обеспечивающая способы восстановления после возникновения событий, прерывающих нормальный поток управления (переполнение, void-вызовы и т.п.) . `Try … Except …`

Последовательность

- Главный принцип организации последовательного кода — упорядочение зависимостей.
- Зависимости должны быть сделаны явными с помощью хороших имен методов, списков параметров, комментариев и вспомогательных переменных.
- Если порядковые зависимости в коде отсутствуют, старайтесь размещать взаимосвязанные выражения как можно ближе друг к другу.

Последовательность

- Главный принцип организации последовательного кода — упорядочение зависимостей.
- Зависимости должны быть сделаны явными с помощью хороших имен методов, списков параметров, комментариев и вспомогательных переменных.
- Если порядковые зависимости в коде отсутствуют, старайтесь размещать взаимосвязанные выражения как можно ближе друг к другу.

Программа должна быть написана так, чтобы ее можно было читать сверху вниз, а не перескакивая с места на место.

Последовательность

Неудачный код (C++)

```
MarketingData marketingData;
SalesData salesData;
TravelData travelData;

travelData.ComputeQuarterly();
salesData.ComputeQuarterly();
marketingData.ComputeQuarterly();

salesData.ComputeAnnual();
marketingData.ComputeAnnual();
travelData.ComputeAnnual();

salesData.Print();
travelData.Print();
marketingData.Print();
```

Как рассчитывается marketingData?

Последовательность

Хороший последовательный код (C++)

```
MarketingData marketingData;
marketingData.ComputeQuarterly();
marketingData.ComputeAnnual();
marketingData.Print();

SalesData salesData;
salesData.ComputeQuarterly();
salesData.ComputeAnnual();
salesData.Print();

TravelData travelData;
travelData.ComputeQuarterly();
travelData.ComputeAnnual();
travelData.Print();
```

- Упоминания каждой переменной располагаются вместе — они «локализованы».
- Код можно разбить на отдельные методы для данных по маркетингу, продажам и поездкам.

Выбор

Рекомендации:

- Нужно следить за порядком блоков *if* и *else*, особенно если они обрабатывают множество ошибок. Убедитесь, что номинальный вариант прослеживается ясно.
- Для последовательностей *if … then …else* и операторов *case* выбирайте порядок, позволяющий улучшить читабельность.
- Для перехвата ошибок используйте блок по умолчанию в операторе *case* или последний блок *else* в цепочке операторов *if … then … else*.

Циклы

Рекомендации:

- Циклы сложны для понимания, поэтому они должны быть максимально простыми. Следует избегать экзотических видов циклов, минимизировать вложенность, делать очевидными входы и выходы цикла.
- Индексы цикла часто употребляются неправильно. Называйте их понятно и используйте только с одной целью.