Технологии программирования

Этапы разработки программного обеспечения:

1) Анализ требований.
2) Проектирование.
3) Кодирование.
4) Тестирование и отладка.
5) Документирование.
6) Сопровождение.

Структурное программирование

I. **Модульность** – разбиение программы на части (модули), которые можно компилировать автономно, отдельно от других частей.

   Модули должны как можно **меньше** быть связанными друг с другом.

   Если компилируется (тестируется) один модуль, то о других ему достаточно «знать» их вход и выход.

Принципы модульного программирования:

1) Большие программы следует разбить на малые независимые подпрограммы.

2) Модуль должен иметь одну точку входа и одну точку выхода.

3) Замена общей памяти на дополнительные параметры модулей.

a, b

M1
a=7

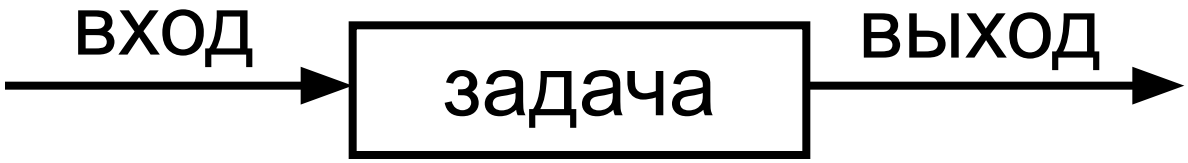
M2
a=a+6

M1(x, y)
x=7

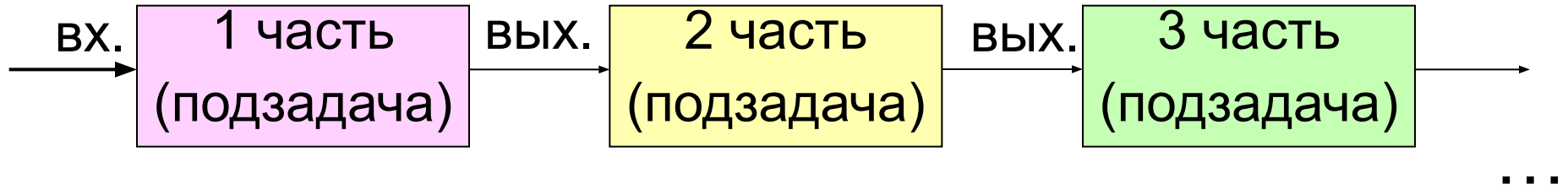
M2(z, v)
z=**z**+6

II. Проектирование **«сверху-вниз»**:

1)

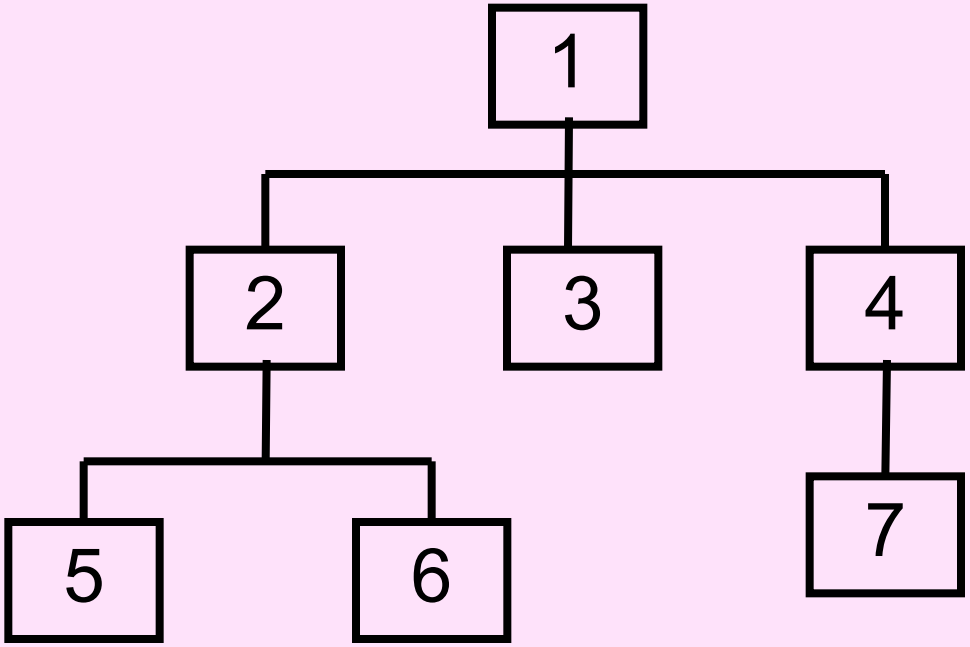


2)

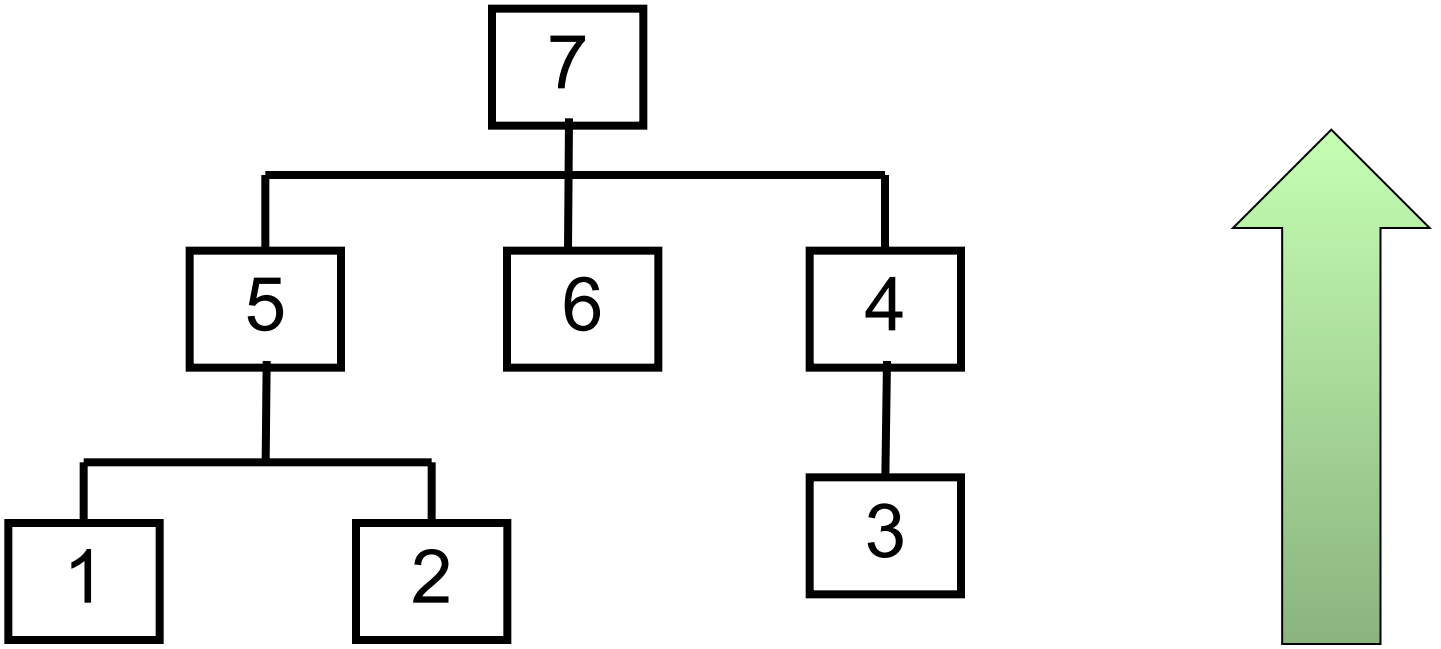


Внутри модуля связи должны быть максимальными, а между модулями – минимальными.

Проектирование программ путем последовательного разбиения большой задачи на меньшие подзадачи, рассматриваемые порознь, соответствует нисходящему («**сверху-вниз**») проектированию.



Восходящее «снизу-вверх»:



Принципы проектирования программ «**сверху-вниз**»:

1) Последовательная **декомпозиция** большой задачи на более мелкие подзадачи (модули);
2) **Спецификация** интерфейсов: описание входа и выхода каждого модуля;
3) Проектирование модулей верхнего уровня производится **без детализации** описания модулей нижних уровней.

III. Программирование **без «GOTO»**.

Для написания программы достаточно использовать:

-операторы присваивания;

-последовательности;

-ветвления (if … then … else …);

- цикл While.

Подпрограммы

Использование подпрограмм позволяет:

- сократить листинг программы;
- улучшить читаемость программы;
- упрощает отладку программы;
- позволяет вызывать подпрограмму из разных участков программы;
- облегчает модификацию программы;
- облегчает независимую отладку блоков программы.

Использование подпрограмм требует согласования параметров подпрограмм.

Основные алгоритмические конструкции подпрограмм:

- **следование** (последовательность);
- **ветвление** (условный оператор);
- **повторение** (циклы).

Подпрограммы вызываются из основной программы **по имени**.

Функции бывают **стандартные** и **определенные пользователем**.

Функция в отличие от процедуры возвращает в точку вызова ***скалярное значение***.

Передача данных из главной программы в подпрограмму и возврат результата осуществляется с помощью **параметров**.

Параметры процедур и функций: **формальные** и **фактические**.

Объектно-ориентированное программирование

Весь мир – это совокупность взаимодействующих объектов.

Объект характеризуется **свойствами** и **поведением**.

Множество объектов, имеющих одинаковое поведение и структуру, образуют **класс объектов**.

Классы – это абстракции, *описывающие* объекты.

Животные

Медведи

- Бурые медведи
- Белые медведи

Собаки

- Колли
- Болонки
- Динго

Коты

- Персы
- Сиамские

Низшие в иерархии классы обладают свойствами (общей структурой) и поведением своих предков, а также имеют и <u>свои специфические</u> свойства и поведение.

Наследование: объекты нижних уровней (подклассов) наследуют структуру и поведение верхних (классов).

Пусть **A** – базовый класс, **B** – его подкласс. Тогда:

- общие для классов **A** и **B** структуры данных и методы могут быть определены только в классе ; **A**
- переменные и методы класса **A** могут быть использованы объектами класса **B** без их повторного определения в **B**.

Класс – это тип данных.

Объект – это экземпляр (конкретный представитель) класса.

Описание класса включает:

- данные (переменные, память под значения этого типа данных);
- методы (функции, процедуры, задающие поведение).

Метод в ООП – **процедура, реализующая действия (операции) над объектом**.

Пример 1.

Класс – «Очередь в магазин»: Очередь_в_маг;

Данные – массив элементов типа «Человек»

Методы – Обслужить («голову») очереди;

Поставить («в хвост») очереди.

Человек Петров; Очередь_в_маг МояОчередь;

тип класс — объект — тип класс — объект

Петров.Кушать;

МояОчередь.Обслужить;

МояОчередь.Поставить(Петров);

Пример 2. Пусть Ivankov – объект класса Student, Name – переменная (свойство объектов) этого класса,
work() – метод класса.
Тогда объектно-ориентированной программой, устанавливающей имя студента и применяющей этот метод, является:

```
Student Ivankov;

Ivankov.Name = “Vasily”;

Ivankov.work();
```

Пример 3. Пусть a – объект класса k, setval (int x) – метод, задающий значение, указанное параметром x, объектам этого класса. Тогда программа, устанавливающая значение **7** объекту a, имеет вид:

```
a.setval(7)
```

Инкапсуляция означает возможность отделения интерфейса спецификации методов от их реализации.

Полиморфизм означает возможность использования разных функций с одним и тем же именем.

Основные концепции объектно-ориентированного подхода:

1. **Класс.**
2. **Объект.**
3. **Метод.**
4. **Свойство.**
5. **Событие.**
6. **Инкапсуляция.**
7. **Полиморфизм.**
8. **Наследование.**

Пример. Класс **Cat** – «Кот». Класс содержит приватную переменную Age (возраст), открытые методы доступа к этой переменной SetAge и GetAge, а также открытый метод кошачьего мяуканья Meow. Вся программа:

```
#include<iostream.h>                    //подключение библиотеки ввода-вывода

class Cat                               //начало объявления класса
{public:
   int GetAge(void);                    //методы класса – открытый доступ
   void SetAge(int x);
   void Meow(void);
 private:
   int Age;                             //переменная класса – закрытый доступ для не членов класса
};                                      //конец  объявления класса
```

```
int Cat::GetAge(void)        //определение (реализация) метода GetAge,
 {return Age;};      // возвращающего значение приватной (закрытой)
                                                    переменной Age
void Cat::SetAge(int x)       //определение (реализация) метода SetAge,
  {Age = x;};                          //устанавливающего значение
                                                    переменной Age:
void Cat::Meow(void)          //определение (реализация) метода Meow
{cout<<"Miay!!!!!!"; };
```

//Наша задача – создать кота, установить его возраст, заставить его мяукнуть, узнать его возраст и еще раз заставить мяукнуть:

```
int main()
{Cat Murka,Barsik;
 Murka.SetAge(5);
 Barsik.SetAge(2);
 Murka.Meow();
 cout<<"Murke - "<<Murka.GetAge()<<" let"<<endl;
 Murka.Meow();
 Barsik.Meow();
 getch();
 return 0; }
```