

A word cloud of various programming languages is visible in the background, including Assembly, C, PHP, JavaScript, Lua, Scheme, Julia, Rust, Haskell, AppleScript, Bash, Pascal, and others. The words are in different colors and sizes, creating a textured background.

ДОП №2

Про языки программирования

Ассемблер x86

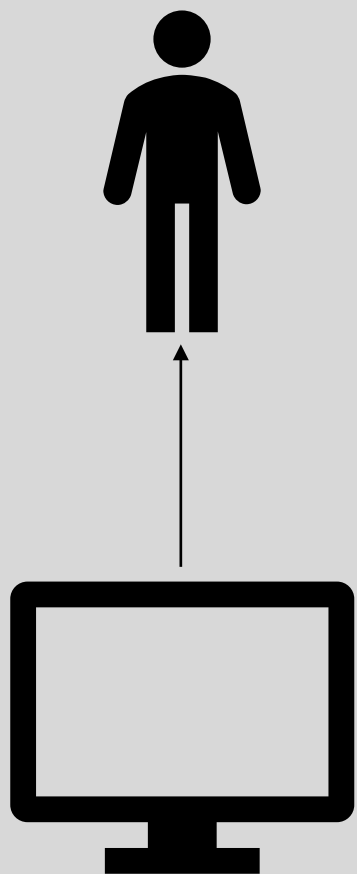
```
C:\NASM\LABELS\tasm\label.asm
        assume cs:cod,ds:dat
;сегмент стека
stk      segment stack
        db 256 dup('<*>')
stk      ends
;сегмент данных
dat      segment
CR       equ 13 ;код возврата каретки
LF       = 10  ;код перехода на новую строку
msgb     label byte
msgw     label word
        db 07h,07h ;код звукового сигнала
msgw1    label word
        db '--Hello '
msgb1    label byte
        db 'World--$'
dat      ends
;сегмент кода
cod      segment
start:   nov ax,dat ;занесем сегментный адрес данных
        nov ds,ax  ;в DS
        nov ah,09h ;функция DOS вывода строки
        lea dx,msgb ;в DX адрес начала строки
        int 21h    ;вызвали функцию DOS
        nov msgb,CR ;поместили код CR вместо 07h
        nov msgb+1,LF ;поместили код LF вместо 07h
        int 21h
        nov msgw,2020h;поместили два пробела
        int 21h
        nov msgw,0d0ah ;поместили CR LF (hex код)
        nov msgw1,2a2ah ;поместили две * вместо --
        int 21h
        nov msgb1,'w' ;заменяли W на w
        nov msgb1+1,'0' ;заменяли 0 на o
        int 21h
        nov ax,4c00h
        int 21h
cod      ends
end start
```

```
.286
TEXT segment
org 100h
```

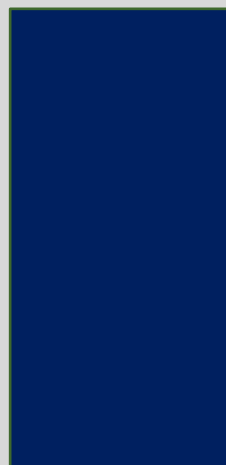
```
Open_file proc
mov ax, 3D02h
mov dx, 1Eh
int 21h
mov Handle, ax
mov bx, ax
ret
```

```
Handle dw 0FFFFh
Open_file endp
```

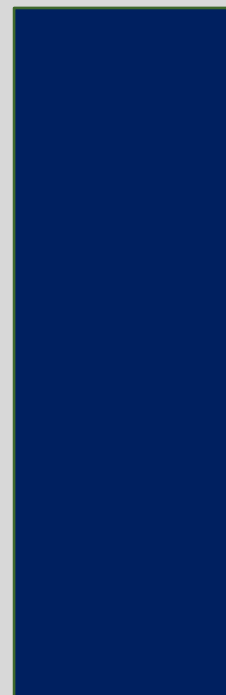
УДОБСТВО



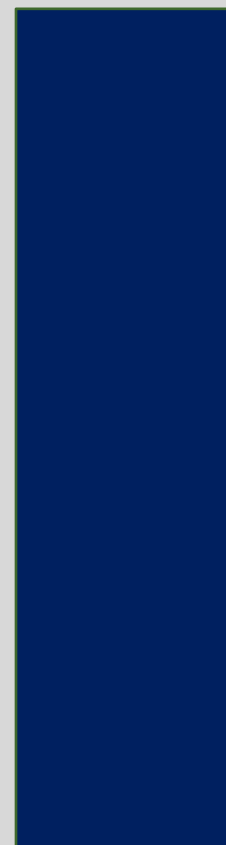
Ассемблер



C/C++



Python, R



Самый низкий уровень абстракций

Assembly (dos)

```
TEXT segment
org 100h

start:
mov ah,9
mov dx, offset message
int 21h
int 20h

message db 'Hello, World!$'
TEXT ends
end start
```

Оперируем «байтами».
Полный контроль над
памятью компьютера.

Повышаем уровень абстракций

Java

```
public class HelloWorld {  
  
    public static void main(String[] args) {  
        System.out.println("Hello, world!");  
    }  
  
}
```

C++

```
#include <iostream>  
using namespace std;  
  
int main(int argc, char* argv[]) {  
    cout << "Hello World!";  
    return 0;  
}
```

И на самом высоком уровне

Python

```
print("Hello, World")
```

Javascript

```
alert("Hello World!");
```

Выбор между скоростью и удобством

Писать дольше, сложнее,
дороже



Работает быстрее

Писать быстрее, проще и
чаще дешевле

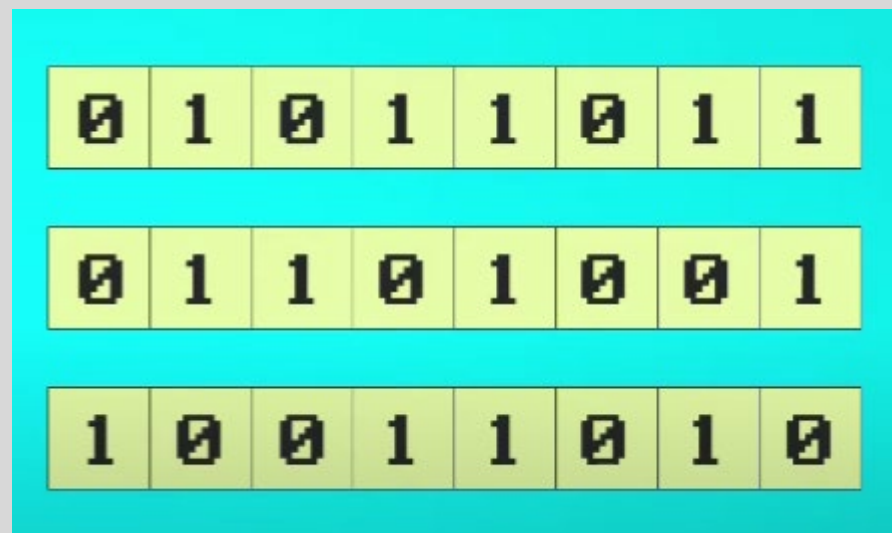


Работает медленнее

```
#include <iostream>
using namespace std;

int main(int argc, char* argv[]) {
    cout << "Hello World!";
    return 0;
}
```

Машинный набор
инструкций



Трансляторы



Компиляторы

Целиком анализирует программу и переводит её с исходного языка в эквивалентную программу на машинном языке, после чего она может быть запущена и выполнена



Интерпретаторы



Покомандно анализирует программу по мере её поступления и сразу же выполняет, не переводя её в аналогичную программу на машинном языке (переводит по мере поступления в машинный код, а не сразу целиком)



Компиляторы

Преимущества компилятора

- Программный код уже переведен в машинный, и, следовательно, требуется меньше времени на его исполнение.
- Файлы .exe выполняются быстрее, чем исходный код. Объектные программы сохраняются и могут быть запущены в любое время.
- Объектные программы пользователю сложнее изменить, чем исходный код.
- Компилятор проверяет исходный код на наличие синтаксических ошибок во время компиляции.

Недостатки компилятора

- Поскольку переводится вся программа, она использует гораздо больше памяти компьютера.
- При работе с компилятором невозможно изменить программу, не вернувшись к исходному коду.
- Необходимо создавать объектную программу перед окончательным исполняемым файлом. Это может занять много времени.
- Исходный код должен быть на 100% верным для создания исполняемого файла.



Интерпретаторы

Преимущества интерпретатора

- Интерпретатор значительно облегчает работу с исходным кодом.
- Он переводит по одной инструкции за раз, поэтому использует минимальный объем памяти.
- Интерпретатор может связать сообщения об ошибках с выполняемой инструкцией, что может оказаться полезным в процессе отладки.

Недостатки интерпретатора

- Каждый раз, когда программа выполняется, тратится время на интерпретацию, из-за чего затягивается время исполнения.
- Интерпретируемые программы могут выполняться только на компьютерах, на которых имеются соответствующие интерпретаторы.

Синтаксис – набор правил, по которому необходимо писать код, чтобы его понял транслятор данного языка. Синтаксис – то, что ожидает увидеть на входе транслятор. Обычно он описан в документации языка.

Сравнение языков: парадигма, типизация

Парадигма – стиль написания кода
(как и что реализовать?)

Императивный
стиль

Пошагово расписываем как решить задачу в виде последовательных действий, которые должны привести к результатам

Декларативный
стиль

Не расписываем шаги, а пишем какой результат нам нужен

```
$x = 5;  
$y = 10;  
function sum($x, $y) {  
    return $x + $y;  
}  
$result = sum($x, $y);
```

C++, Java, Python и
большинство
других

```
"SELECT * FROM `users`  
WHERE id = 1 AND activate = 1  
ORDER BY lastname ASC";
```

SQL

Императивный СТИЛЬ

Процедурный

```
function sum(a, b) {  
    return a + b;  
}  
function mul(a, b) {  
    return a * b;  
}  
function getResult(a, b) {  
    return sum(a, b) + mul(a, b);  
}
```

ООП

```
class Calc {  
    sum(a, b) {  
        return a + b;  
    }  
    mul(a, b) {  
        return a * b;  
    }  
    getResult(a, b) {  
        return this.sum(a, b) +  
            this.mul(a, b);  
    }  
}
```

Типизация

Явная/неявная

```
int a = 5;  
String b = "string";  
double c = 7.5;
```

```
let a = 5;  
$b = "string";  
c = 7.5
```

Статическая/ динамическая

Статическая – проверка и установка типов данных проходит на этапе компиляции, а в динамической – на этапе выполнения программы.

Это одна из причин, почему языки со статической могут работать быстрее.

Типизация

Сильная/слабая (строгая/нестрогая)

Сильная типизация не позволит вам смешивать данные разных типов.

```
int a = 5;  
double b = 7.5;  
a = a + b; // error
```

Компилятор языков со слабой типизации позволяет это делать, автоматически выполняя преобразование типов, что может приводить к неоднозначным результатам.

```
int a = 5;  
double b = 7.5;  
a = a + b; // 12
```


Язык R?

- Попробуем описать его

Язык R?

- Интерпретируемый
- Императивный стиль
- Неявная типизация (тип переменной задавать не нужно)
- Динамическая типизация (тип присваивается на этапе работы программы)
- Строгая типизация (не преобразовывает данные автоматически к нужному типу)