Лекция 3 Стандартная библиотека языка С

План курса «Встраиваемые микропроцессорные системы»:

Лекция 1: Введение. Язык программирования С

Лекция 2: Язык программирования С, применение для встраиваемых систем

Лекция 3: Стандартная библиотека языка С

Лекция 4: Ядро ARM Cortex-M3. Микроконтроллер Миландр К1986ВЕ92QI

Лекция 5: Этапы разработки микропроцессорных систем

Лекция 6: Разработка и отладка программ для встраиваемых систем

Лекция 7: Архитектура программного обеспечения

Лекция 8: Периферийные модули: Timer, DMA, ADC, DAC

Лекция 9: Периферийные модули: CAN, USB, Ethernet, SDIO

Стандартная библиотека C

Заголовочный файл	Назначение	Пример
`#include <ctype.h>`	Работа с символами	`int tst = isupper(ch);`
`#include <math.h>`	Математические операции	`float a = exp(1.0);`
`#include <stdlib.h>`	Преобразование типов, выделение памяти	`int n = atoi("42");`
`#include <stdio.h>`	Работа со стандартным потоком ввода/вывода, работа с файлами, форматированный ввод/вывод	`printf("Hello, world!\r\n");`
`#include <string.h>`	Строковые операции	`int eq = strcmp(s, "run");`
`#include <stdint.h>`	Целочисленные типы	`uint8_t a = 0;` `int32_t i;`

Существую и другие заголовочные файлы.

Работа с символами

```
#include <ctype.h>

int tst;
char ch = 'D';

tst = isalpha(ch);/* Отображаемый символ, tst = 1 */
tst = isdigit(ch);/* Цифра 0 - 9, tst = 0 */
tst = isupper(ch);/* Верхний регистр, tst = 1 */
tst = islower(ch);/* Нижний регистр, tst = 0 */

ch = tolower(ch);/* Перевести в нижний регистр, ch = 'd' */
ch = toupper(ch);/* Перевести в верхний регистр, ch = 'D' */
```

Математические операции

```
#include <math.h>
/* Существует вариант библиотеки с суффиксом f(mathf) для работы с
числами float.
Функции из этой библиотеки также имеют суффикс f, например, sinf */

double a;

a = sin(1.5);   /* Вычисление синуса */
a = asin(1.0);  /* Вычисление арксинуса */
a = ceil(0.99); /* Округление до ближайшего большего целого числа */
a = floor(0.99);/* Округление до ближайшего меньшего целого числа */
a = exp(1.0);   /* Вычисление экспоненты */
a = log(10.0);  /* Вычисление натурального логарифма */
```

Преобразование типов, выделение памяти

```c
#include <stdlib.h>

int a = abs(-10);    /* c = 10 */
int b = atoi("100");/* d = 100 */

srand(42);        /* Инициализация генератора псевдослучайных чисел */
int n = rand(); /* Генератор псевдослучайных чисел */

int *buf = malloc(100); /* Выделение памяти из кучи */
free(buf);              /* Освобождение памяти в кучу */
```

Форматированный вывод

```c
#include <stdio.h>

printf("Hello, world!\n");

int i = 2, j = 3;
/* Вывод целых чисел со знаком */
printf("i=%d j=%d\n", i, j); /* "i=2 j=3" */

i = 15;
/* Вывод целых чисел в шестнадцатеричном
представлении */
printf("i=%x j=%x\n", i, j); /* "i=f j=3" */

/* Вывод чисел с плавающей запятой */
float ch1 = 10.1, ch2 = 12.3;
printf("ch1=%f, ch2=%.1f\n", ch1, ch2);/* "ch1=10.100000, ch2=12.3" */

/* Вывод строки */
char *msg = "overcurrent";
printf("Fail: %s\n", msg); /* "Fail: overcurrent" */
```

Форматированный ввод

```c
#include <stdio.h>
unsigned int a;
scanf("%u", &a); /* Ввод целого числа без знака*/

int b, c;
scanf("%d %d", &b, &c); /* Ввод двух целых чисел со знаком разделенных пробелом */
scanf("%d, %d", &b, &c); /* Ввод двух целых чисел со знаком разделенных запятой */

float x;
scanf("%f", &x); /* Ввода числа с плавающей запятой */

char buf[20 + 1];
scanf("%s", buf); /* Ввод до первого символа пробела, табуляции или новой строки */
scanf("%20s", buf); /* Более безопасный вариант */
gets(buf); /* Ввод строки до символа новой строки */
fgets(buf, sizeof(buf), stdin); /* Более безопасный вариант */

int h, m, s;
/* Ожидание строки вида "Time: 12 h 00 m 00 s" */
if (scanf("Time: %d h %d m %d s", &h, &m, &s) == 3)
    ...
```

Чтение из файла

```c
#include <stdio.h>

/* Объявление файлового указателя */
FILE *fp;
/* Открытие файла test.txt на чтение ("r").
Если файла не существует, то fopen возвратит NULL. */
if ((fp = fopen("test.txt", "r")) == NULL)
{
    // Обработка ошибки
}
/* Объявление буфера */
char buf[100];
/* Чтение файла по строчно */
while (fgets(buf, sizeof(buf), fp))
    printf("%s\n", buf); /* Печать строки */
/* Закрытие файла */
fclose(fp);
```

Запись в файл

```c
#include <stdio.h>

/* Объявление файлового указателя */
FILE *fp;
/* Открытие файла test.txt на запись ("w").
Открытие файла на запись в режиме "w" стирает его содержимое.
Для добавление данных в конце файле следует использовать флаг "a" */
if ((fp = fopen("test.txt", "w")) == NULL)
{
    // Обработка ошибки
}
/* Функция fprintf аналогична функции printf, только первым
аргументом функции является файловый указатель. */
fprintf(fp, "Hello, world\n");
/* Закрытие файла */
fclose(fp);
```

Строковые операции

```c
#include <string.h>

char msg[] = "Hello";
int len;
len = strlen(msg);      /* len=5 без символа \0*/
len = sizeof(msg);      /* len=6 вместе с \0 */

char cmd[100];
scanf("%s", cmd);

if (strcmp(cmd, "run") == 0) /* Сравнение строк */
{
     run();
}

char buf[256];
strcpy(buf, cmd); /* Копирование cmd в buf */
```

Целочисленные типы

```c
#include <stdint.h>

uint8_t i; /* Беззнаковое целое размером 8 бит */
int16_t j; /* Целое со знаком размером 16 бит */
int32_t k; /* Целое со знаком размером 32 бита */
uint64_t l; /* Беззнаковое целое размером 64 бита */
```

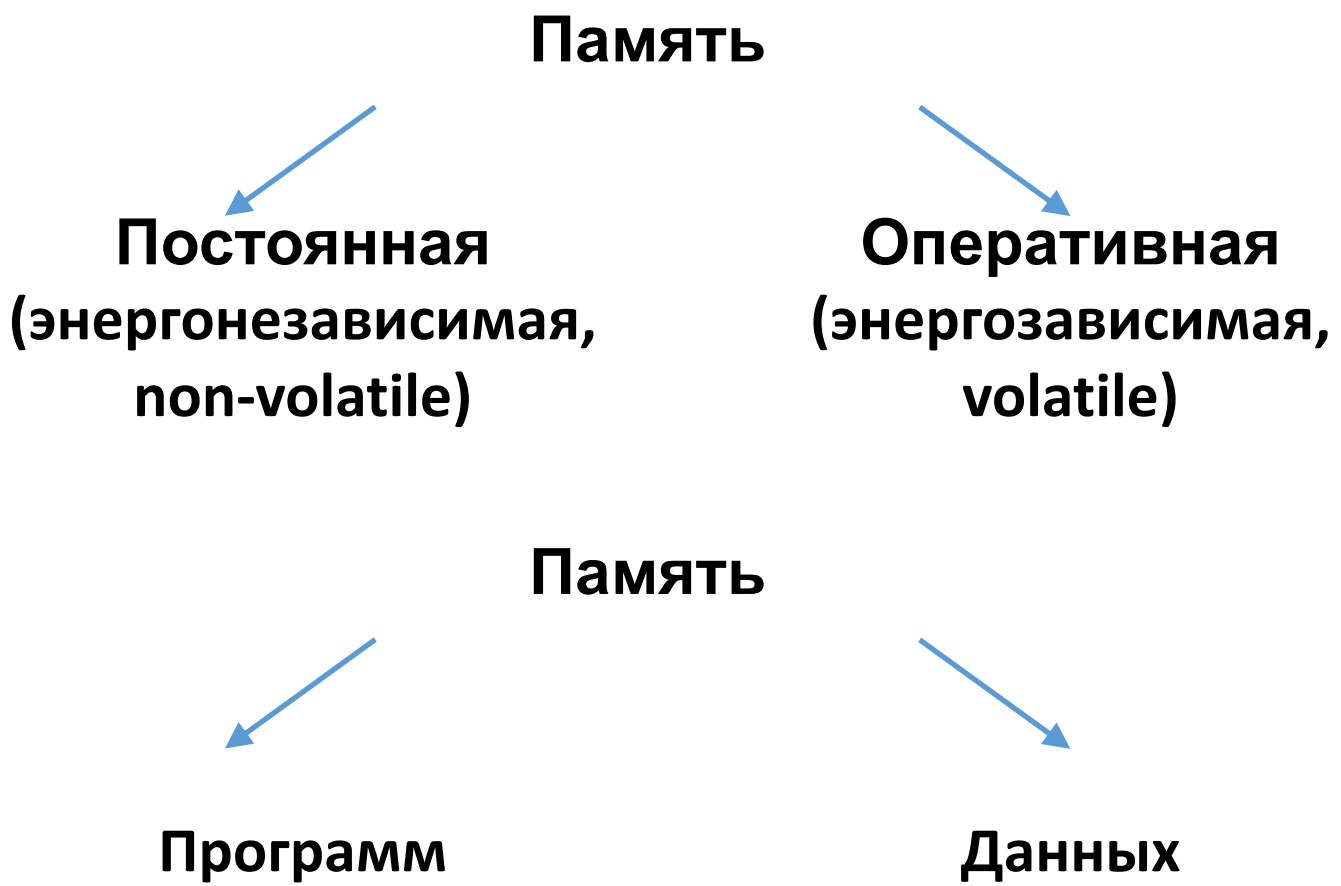
Получение аргументов командной строки

```c
/* Для программ на ПК */
#include <stdio.h>
int main(int argc, char* argv[])
{
    /* argc - количество аргументов */
    /* argv - массив с указателями на аргументы */
    /* argv[0] - имя исполняемого файла */
    int i;
    for (i = 0; i < argc; i++)
        printf("argv[%d]: %s\n", i, argv[i]);

    return 0;
}
```

```
PS .\test.exe arg1 arg2
argv[0]: C:\temp\test.exe
argv[1]: arg1
argv[2]: arg2
```

Классификация памяти



Постоянная память (ПЗУ преимущественно используется как память программ, но также для данных – константы. Оперативная память (ОЗУ) преимущественно используется как память данных, но в высокопроизводительных системах и как память программ.

Модель памяти в языке С

Память данных

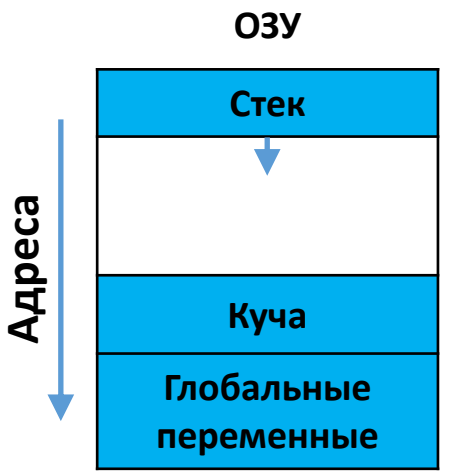
Статическая

выделение при компоновке,
глобальные переменные,
время жизни – время работы программы

```
int a;
int b = 42;
void main()
{
    static int j;
    …
}
```

Динамическая

выделение в процессе работы программы



Автоматическая

стековые переменные,
время жизни – внутри фигурных скобок

```
void main()
{
    int d;
    char e[100];
    …
}
```

Ручная

переменные из кучи,
время жизни – до вызова функции free

```
void main()
{
    char *p = malloc(100);
    …
    free(p);
}
```

Динамическая и статическая память

Статическая память

Выделение памяти при компоновке (Link Time).

+ Легко управлять. Быстрая инициализация.

+ Детерминированное поведение. Атомарность.

- Фиксированный размер.

Оптимально когда нужно использовать ресурсы одновременно.

```
#define ARR_SIZE 32
char x[ARR_SIZE]; // создание
char a[ARR_SIZE];
…
x = {…};          // инициализация
a = {…};
…
filter(x, a);     // исполнение
…
```

Динамическая память

Выделение памяти при исполнении (Run Time).

+ Совместное использование общих ресурсов.

+ Память может быть освобождена.

- Сложнее управлять.

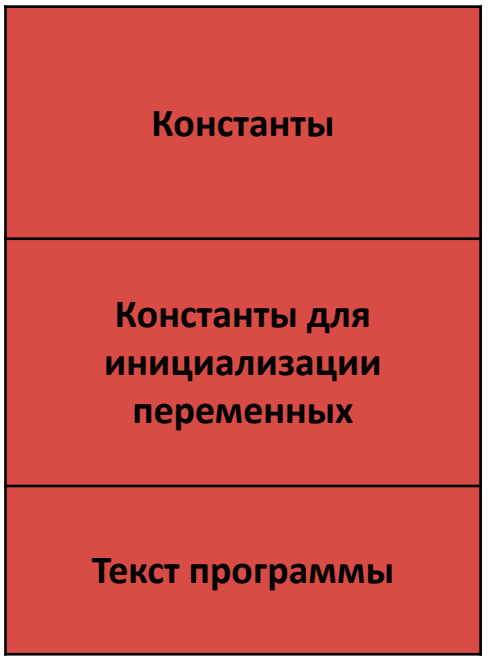
- Недетерминированное поведение. Не атомарность.

Оптимально когда неизвестен необходимый объем памяти или несколько задач использует одни и те же ресурсы.

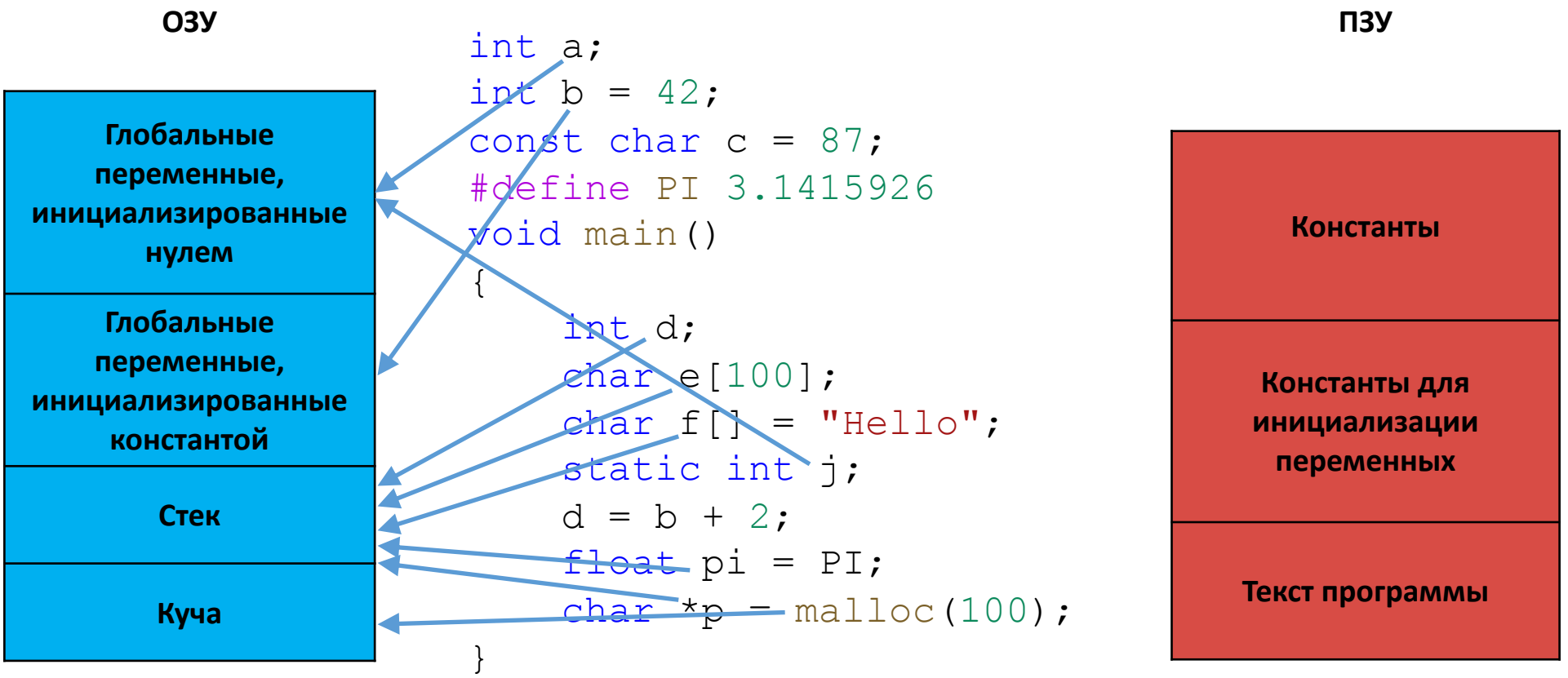
```
#define ARR_SIZE 32
char* x = malloc(ARR_SIZE); // создание
char* a = malloc(ARR_SIZE);
…
x = {…};                    // инициализация
a = {…};
…
filter(x, a);               // исполнение
…
free(x);                    // освобождение
free(a);
```

Модель памяти в языке С

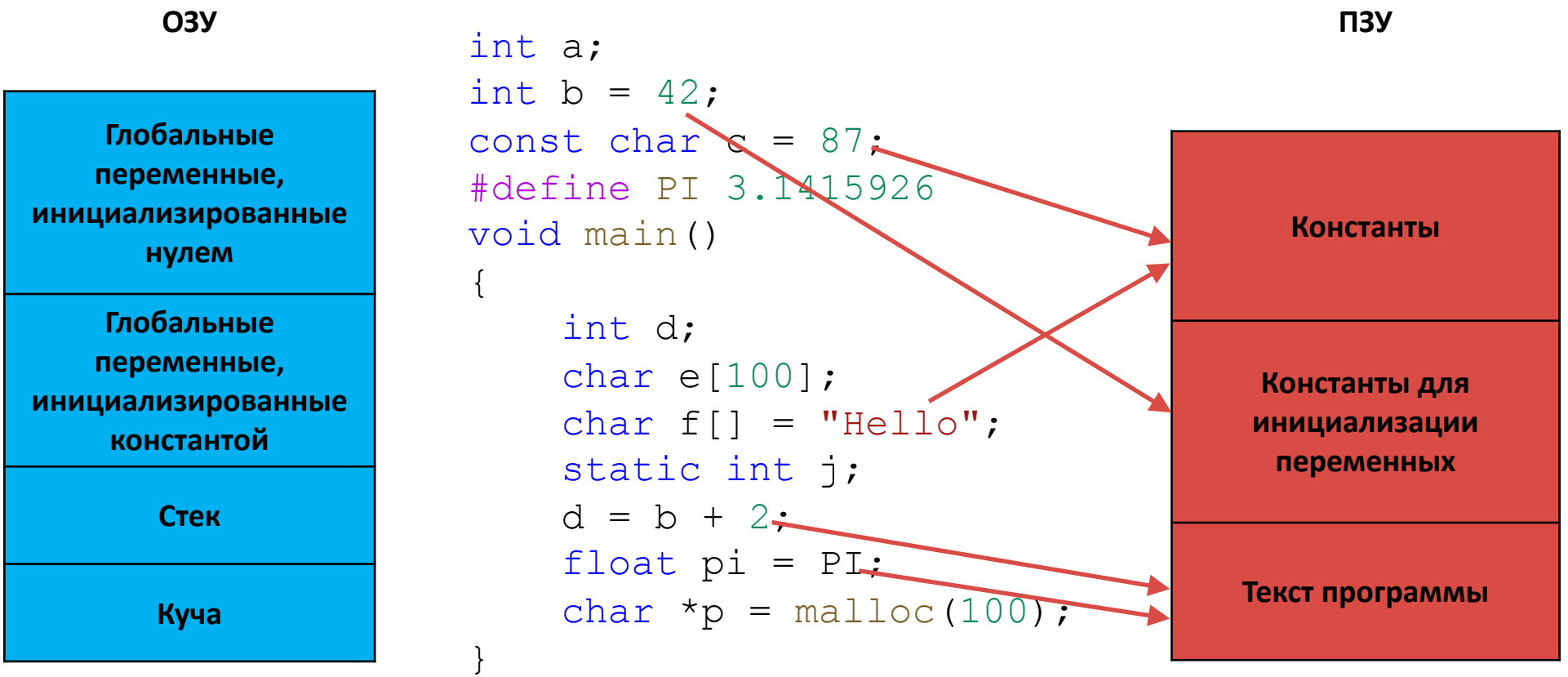

```c
int a;
int b = 42;
const char c = 87;
#define PI 3.1415926
void main()
{
    int d;
    char e[100];
    char f[] = "Hello";
    static int j;
    d = b + 2;
    float pi = PI;
    char *p = malloc(100);
}
```



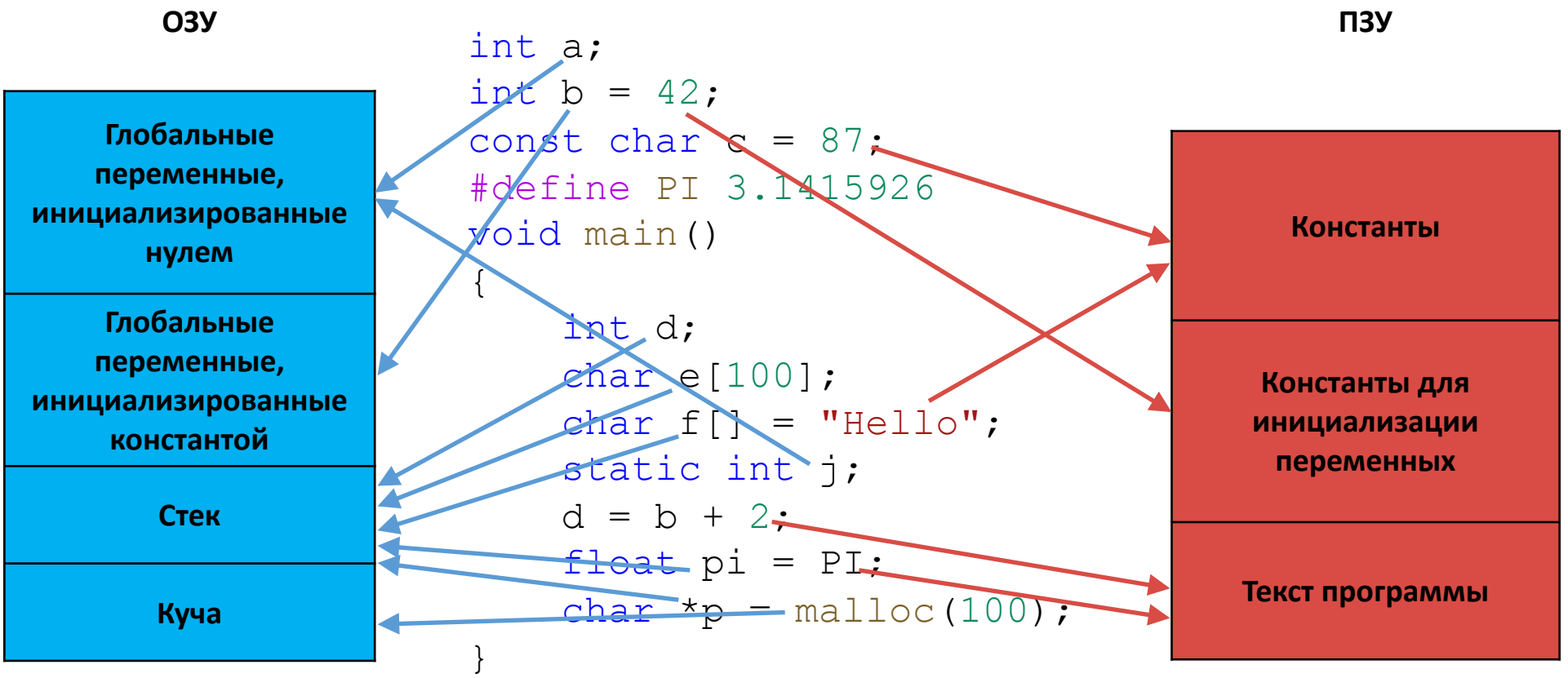
Модель памяти в языке С



Модель памяти в языке С



Модель памяти в языке С



Заключение

- Стандартная библиотека C содержит множество функций, которые позволят не «изобретать велосипед»;
- Важными являются функции форматированного ввода (scanf) и вывода (printf).
- Понимание модели памяти языка C позволяет писать оптимальный код.