Операции и данные C/C++

Алгоритмы
и алгоритмические языки

goo.gl/c8pyqx

Лекция 4, 28 сентября, 2018

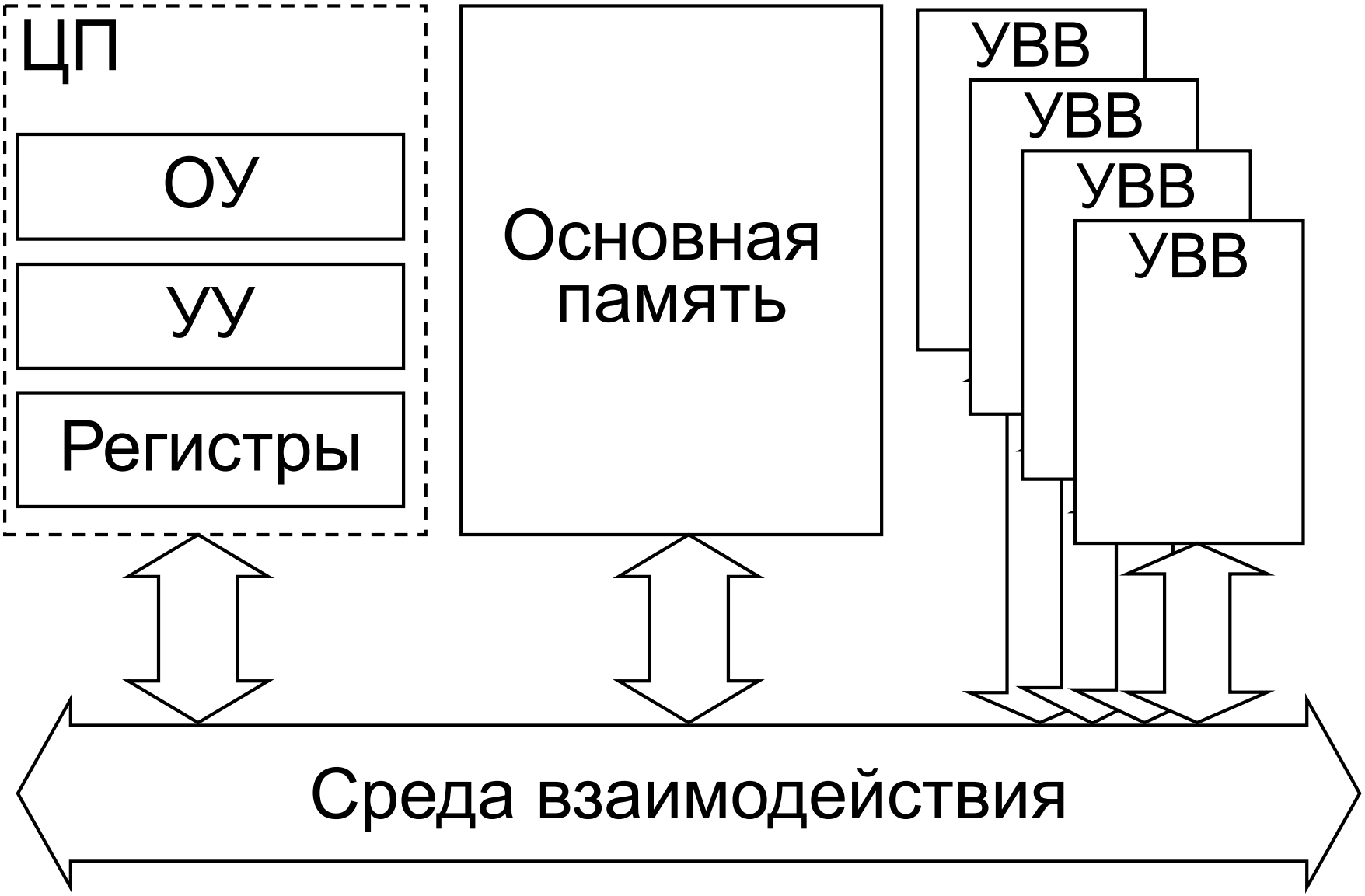
Лектор:

Дмитрий Северов, кафедра информатики 608 КПМ

dseverov@mail.mipt.ru

http://cs.mipt.ru/wp/?page_id=6077

Модель исполнителя



Переменная

Именованная область памяти типизованных значений.

1. Имя (адрес начала),
2. Тип (размер и правила операций),
3. Значение (содержимое памяти)

Эволюция базовых типов данных

- Базовые типы данных
  - Целые (char, int)
  - Вещественные (float, double)
  - Никакой (void)
- Модификаторы типа
  - Без/Знаковые (un/signed)
  - Короткие/Длинные (short, long)
- Спецификаторы класса памяти:
  - Статические/Автоматические (static, auto)
  - Регистровые, Внешние (register, extern)
- Квалификаторы изменчивости:
  - Неизменяемые (const),
  - Изменчивые (volatile), исключаемые из оптимизации

Приоритет операций

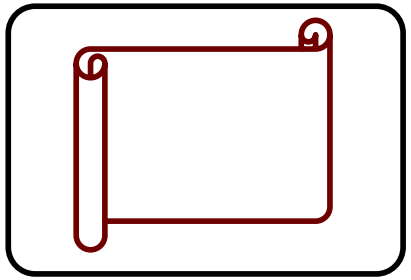
Приоритет и порядок	Обозначение	Назначение
1⇨	::	Разрешения контекста
2⇦	()	Вызов или описание функции
	()	Конструкция значения
	[]	Индекс массива
	->	Косвенная принадлежность
	.	Прямая принадлежность
	++	Инкремент (постфиксная)
	--	Декремент (постфиксная)
3⇨	!	Логическое отрицание
	~	Битовое отрицание (сумма по модулю 2)
	+	Унарный плюс
	-	Унарный минус
	++	Инкремент (префиксная)

Приоритет и порядок	Операция	Назначение
3⇨	--	Декремент (префиксная)
	&	Адрес
	*	Разыменование
	()	Преобразование типа
	sizeof	Размер в байтах
	new	Динамическое выделение памяти
	delete	Динамическое освобождение памяти
4⇨	.*	Разыменование члена
	->*	Косвенное разыменование члена
5⇨	*	Умножение
	/	Деление
	%	Остаток от деления
6⇨	+	Сложение
	-	Вычитание
7⇨	<<	Сдвиг влево
	>>	Сдвиг вправо

Приоритет и порядок	Операция	Назначение
8⇨	<	Меньше
	<=	Меньше или равно
	>=	Больше или равно
	>	Больше
9⇨	==	Равно
	!=	Не равно
10⇨	&	Побитовая конъюнкция
11⇨	^	Побитовое сложение по модулю два
12⇨	|	Побитовая дизъюнкция
13⇨	&&	Конъюнкция
14⇨	||	Дизъюнкция
15⇨	?:	Условное выражение

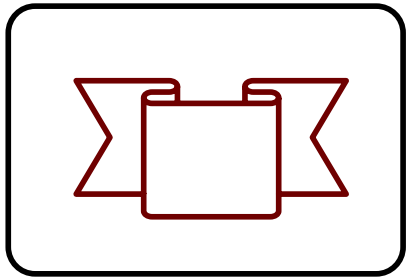
Приоритет и порядок	Операция	Назначение
16⇨	=	Присваивание
	*=	Умножение и присваивание
	/=	Деление и присваивание
	%=	Нахождение остатка и присваивание
	+=	Сложение и присваивание
	-=	Вычитание и присваивание
17⇨	&=	Побитовая конъюнкция и присваивание
	^=	Побитовое сложение по модулю два и присваивание
	|=	Побитовая дизъюнкция и присваивание
	<<=	Сдвиг влево и присваивание
	>>=	Сдвиг вправо и присваивание
18⇨	throw	Генерация исключения
19⇨	,	Объединение двух выражений в одно

Составные типы данных



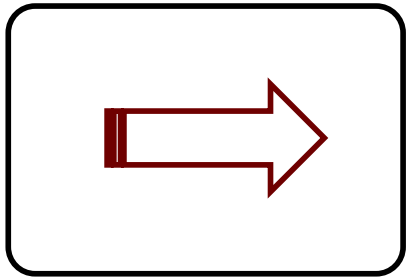
Массивы – наборы нумерованных однородных данных

- Строки



Структуры – наборы именованных разнородных данных

- Объединения
- Перечисления



Указатели – переменные содержащие адрес

- Ссылки

Массивы

- Описание

```
int a[3],b[2][3];
```

- Описание с инициализацией

```
char c[] = {'0','2','4'};
int d[][3] =
 {{1,2,3},{4,5},{6}};
```

- Доступ к элементу

```
a[0] = d[0][1] + 1;
```

Строки

- Одномерные массивы однобайтных целых элементов (символов), содержащие последним элемент со значением ноль.

- ```
  char Str1[] = {'H','e','l','l','o','\0'};
  char Str2[5] = {',',' '};
  char Str3[] = "world";
  ```

- Использование

  ```
  cout << Str1 << Str2 << Str3 << endl;
  ```
  ```

Структуры

- Определение конструкции набора данных:

```
struct My_type {
        type_a Member_1,Member_2;
        type_b Member_3;… } My_object;
```

- Частные случаи:

```
struct Complex { double Re,Im; };
struct Complex A;
struct  { double Re,Im; } B;
```

- Доступ к полям: `B.Re` или `A.Im`

Объединение

совмещение полей — точек зрения

`union` *тег* `{` *список_полей* `}` *имя_объекта*`;`

Перечисление

Набор именованных целых констант

`enum` *тег* `{` *список_имен* `}` *имя_объекта*`;`

Битовые поля

Набор целых полей определённой длины

`struct` *тег* `{` *поле*`:`*число_бит*`;` `}` *имя*`;`

`union` *тег* `{` *поле*`:`*число_бит*`;` `}` *имя*`;`

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

enum {Z=0,X,Y=4};
union {
    int A;
    char B[4];
    struct { int a:4,b:4,c:4,d:4,e:4,f:4,g:4,h:4;} X;
    struct { short int L_word, H_word;} W;
} N;

int main() {
    N.B[0]=Y; N.B[1]=Z; N.B[2]=X; N.B[3]=Z;
    cout << N.W.H_word << " " << N.W.L_word << endl;
    cout << N.A << endl;
    cout << N.X.h << N.X.g << N.X.f << N.X.e
         << N.X.d << N.X.c << N.X.b << N.X.a << endl;
    return 0;
}
```

A								
B	B[0]=Y		B[1]=Z		B[2]=X		B[3]=Z	
X	a	b	c	d	e	f	g	h
W	L_word				H_word			

1 4
65540
00010004

УКАЗАТЕЛИ

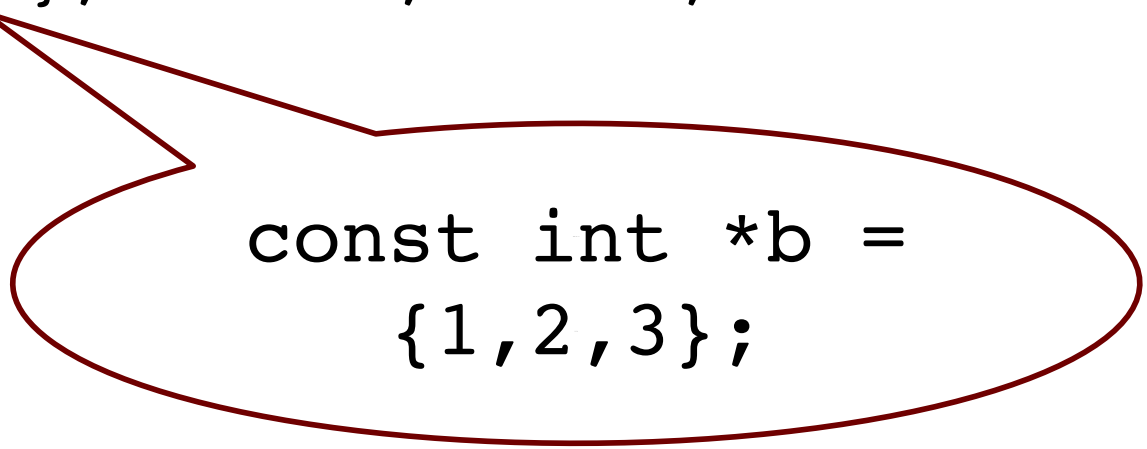
- Переменные, содержащие адрес

- Описание и инициализация

```
int *Pc, **Pd;
int a=5, b[]={1,2,3}, *Pa=&a, *Pb=b;
char *s="String";
```

- Присваивание значения

```
Pd=&Pb; Pc=&b[1];
```



Операции с указателями

```
int n, b, a[10], *pb=&a[0], *pe=&a[9], *pc;
```

- Арифметические

```
n = pe - pb;  // кол-во элементов (не байт!)
pb++;         // &a[1] (адрес следующего
              // элемента но не байта!)
pc = pe – 1;  // &a[8]
pb + pe;      // ОШИБКА !!!
```

- Разыменования

```
b = *pc;      // b = a[8];
```

NB: операции с указателями являются размерными!!!

Операции с указателями

```
struct complex { double Re; double Im; } a, *p;
p = &a;
```

- Разыменования

```
a.Re = 1.;
p->Im = 0.;
```

NB: операции с указателями являются размерными!!!

Универсальные указатели

- `void *P, *Q;`
- Запрещены операции
  - Арифметические
  - Разыменования
- Типичное применение

```
P= malloc(sizeof(double));// P = new double;
Q= calloc(N,sizeof(int)); // Q = new(int[N]);
free(Q);                  // delete(Q);
Q= realloc(Q,M);
```

- Универсальный указатель можно
  - Инициализировать значением указателя любого типа
  - Преобразовать в указатель любого типа

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

int main() {
   const int n= 64;
   char  str[]= "Hello, World!\n";
   void  *p, *ps= str;
   char **pps= (char**)&ps;
   int *a, *pb, *pe;

   cout << str << ps << " " << pps << '' << (**pps) << endl;
   p= malloc(4*n); pb= a= (int*)p; for (int i=0; i<n; i++) a[i] = i;
   cout<< pb << "=pb\t pb+1=" << pb+1 << "\n";
   cout<< pb+2 << ": a[2]= » << *(pb+2) <<  endl;
   free(a); pb=a=NULL;
   cout << pb << endl;

   pb=a=new int[n]; for(int i=0;i<n;i++) *(a+i)=10*i;
   cout << a << "\t" << *(pb+2) <<endl;
   pe=&a[n-1];
   cout <<pe-pb<< endl;
   a=(int*)realloc(a,2*n);
   cout<<"a="<< a <<"\ta[0]="<<*a<<
    "\na[n-1]="<<a[n-1]<<" a[n]="<<a[n]<<endl;
}
```

```
Hello, World!
0x7fff5fbff559 0x7fff5fbff4e0 H
0x100519920=pb pb+1=0x100519924
0x100519928: a[2]= 2
0x0
0x100519920 20
63
a=0x100519920 a[0]=0
a[n-1]=630 a[n]=64
```

Ссылки*)

- Другое имя переменной (псевдоним)
- Описание и инициализация

```
int a=5, &b=a;
                // a и b – разные имена одной
                // и той же области памяти
```

*) только в C++

Тип FILE*

```
typedef struct _iobuf FILE;
```

Или

```
#define FILE struct _iobuf

FILE *a;
FILE* fopen(
        const char* name,
        const char* access
    );
int fclose(FILE* a);
```

```
struct _iobuf {
    char *_ptr;
    int _cnt;
    char *_base;
    int _flag;
    int _file;
    int _charbuf;
    int _bufsiz;
    char
*_tmpfname;  };
```