Структуры данных

Элементарные типы данных

Абстрактные типы данных

Деревья и графы

Элементарные типы данных

Базовые типы

Массивы

Списки

Составные структуры данных

Базовые типы*

* Как правило, эти типы соответствуют готовым аппаратным представлениям

$T_0 = -273\ ^oC$
$\pi = 3{,}1415926535\ \ldots$
$e = -1{,}602176565 \bullet 10^{-19}$ Кл - заряд электрона
$c = 299\ 792\ 458$ м/с - скорость света в вакууме

Целые числа

… -2 -1 0 1 2 …

- Константы

48 060 0x30

- Типы

int char

‘0’ ‘\60’ ‘\x30’

Вещественные числа

3,14 $300 \bullet 10^3$ км/с

- Константы

48. +480.0e-1

- Типы

double float

48.f +.48e2F

ASCII - American National Standard Code for Information Interchange

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
00	nul	soh	stx	etx	eot	enq	ack	bel	bs	ht	nl	vt	np	cr	so	si
10	dle	dcl	dc2	dc3	dc4	nak	syn	etb	can	em	sub	esc	fs	gs	rs	us
20	sp	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
30	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
50	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
60	`	a	b	c	d	e	f	g	h	I	j	k	l	m	n	o
70	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	del

Модификация базовых типов

Со знаком – signed

Без знака – unsigned

Короткое – short

Длинное – long

$$1 = \text{sizeof(char)} \leq \text{sizeof(short)} \leq$$
$$\leq \text{sizeof(int)} \leq \text{sizeof(long)}$$

$$\text{sizeof(float)} \leq \text{sizeof(double)} \leq$$
$$\leq \text{sizeof(long double)}$$

Преобразование типов

- Неявное
- Приведение или Явное

Пример

```
int x,N;
   ((float) x) / N;
```

Типичная ошибка

```
   double s = 1 + 1/2 + 1/3 + 1/4 + 1/5 ;
```

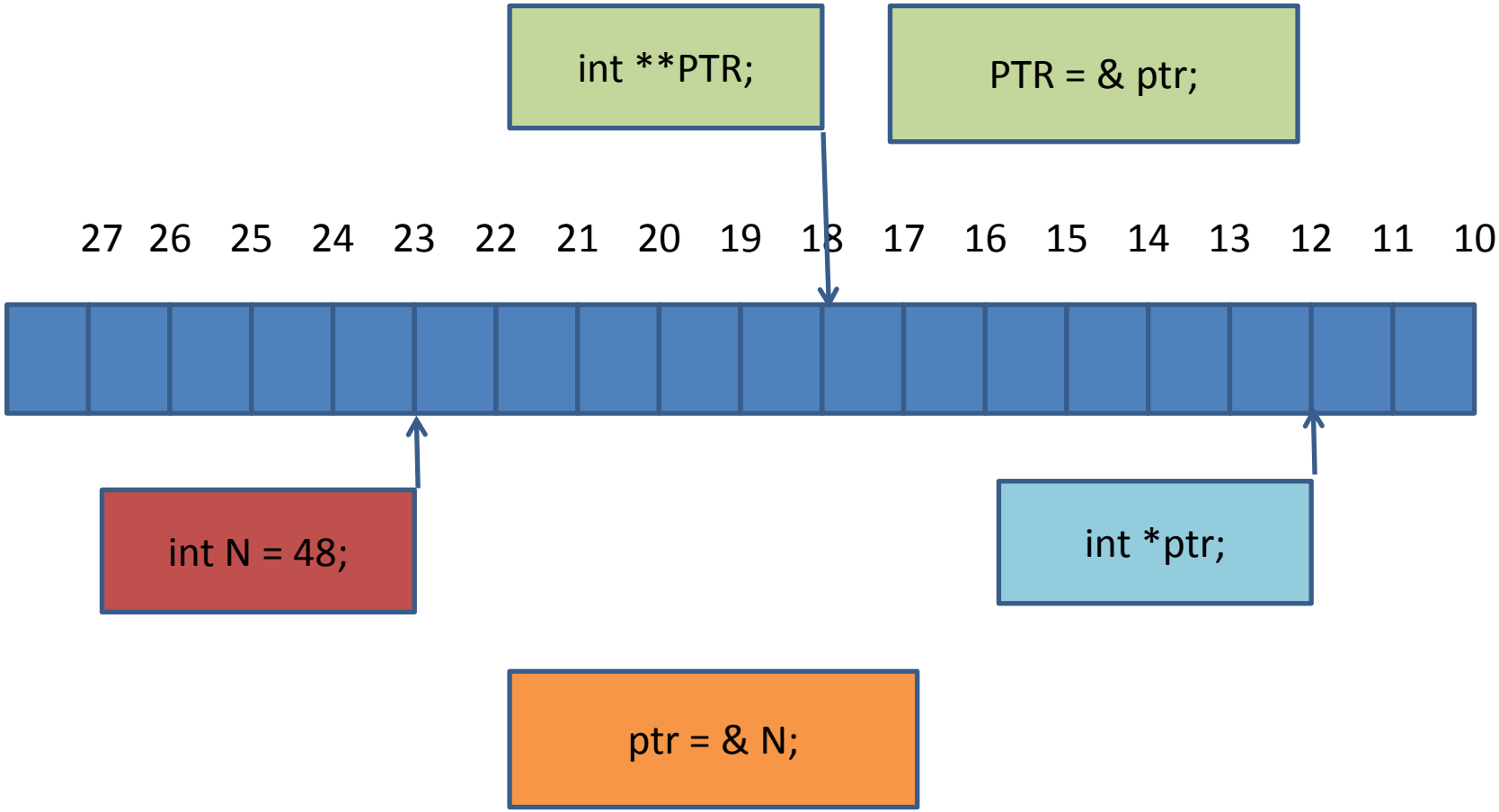
Переменные

- Область памяти для размещения данных , характеризующаяся
  - адресом начала выделенной памяти (имя),
  - размером выделенной памяти (тип),
  - занесенным в эту память двоичным кодом (значением)
- int a; float b; // a,b – будут содержать значения базового типа

  int *A; float *B; // A,B – будут содержать адреса памяти, начиная с которых размещены данные базового типа

  A = &a; b = *A;

  void* c ;


int **PTR;
PTR = & ptr;
27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10
int N = 48;
int *ptr;
ptr = & N;

```
#include <stdio.h>
int **PTR;
char *s="PTR=&ptr=%p, ptr=&N=%p N=*ptr=**PTR=%d\n",
        *s1="&main()=%p &f()=%p &printf()=%p\n";

void f() {
    printf("&PTR=%p &s=%p &s1=%p c=%s",&PTR,s,s1,s1-3); }

int main(int argc, char** argv)
{
   int N=48;
   int *ptr = &N;
   PTR = &ptr;
   printf(s,PTR,ptr,**PTR);
   printf("&argc=%p &argv=%p\n",&argc,&argv);
   printf(s1,&main,&f,&printf);
   f();
   return 0; }
```

Адсес	Имя переменной	Значение
22FF40	ptr	22FF44
22FF44	N	48
22FF48		
22FF4C		
22FF50	argc	
22FF54	argv	
4012D0	f(){…}	
401306	main(){…}	
4022A0	printf(){…}	
404000	s	PTR=&ptr=%p, ptr=&N=%p N=*ptr=**PTR=%d\n
404028	s1	&main()=%p &f()=%p &pririti()=%p\n
4050A0	PTR	22FF40

Сегмент стека

Сегмент программного кода

Сегмент инициализированных данных

Сегмент неинициализированных данных

Определение новых операций с данными

- Все функции имеют список аргументов (параметров)
- Функция может создавать возвращаемое значение
- Функция объявляется путем присвоения ей имени и типа возвращаемого значения
- Функция определяется посредством описания набора операций

Пример

```
#include <stdio.h>
int log2(int);
int main() {
int N;
   for(N=10; N<1000000; N*=10)
                        printf(“%i %d\n”,N,log2(N));
}
int log2(int N) {
int i;
   for(i=0; N>0; i++, N /= 2);
   return i;
}
```

Группировка данных

Массивы

- Фиксированный набор однотипных данных:

int a[10];

Доступ к элементам

```
for(int i=0;i<10;i++) a[i]=0;
```

Альтернатива

```
for(int i=0;i<10;i++) *(a+i)=0;
```

Возможный вариант

```
const int N=10;
int a[N];
```

Структуры

- Конгломерат переменных базовых типов, например:

```
struct complex
  { double re; double im; };
```

Объявление

```
struct complex z;
```

Доступ к членам

```
z.re = 1.; z.im = 1.;
```

Функции

```
complex sum(complex a,
  complex b) { complex c;
  c.re=a.re+b.re;
  c.im=a.im+b.im; return c; }
```

КОНГЛОМЕРАТ – механическое соединение чего-либо разнородного, беспорядочная смесь.

Словарь иностранных слов

Решето Эратосфена

```
#include <stdio.h>
const N=1000;
int main() {  int i, j, k, a[N];
   for(i=2; i<N; i++) a[i]=1;
   for(i=2; i<N; i++) if(a[i]) for(j=i;(k=i*j)<N;j++) a[k]=0;
   for(i=2; i<N; i++) if(a[i]) printf("%d ",i); printf("\n");
   return 0;
}
```

Динамическое выделение памяти массиву

```
#include <stdio.h>
int main(int argc, char* argv[]) {
    int i, j, k, N=atoi(argv[1]), *a=malloc(sizeof(int)*N);
    for(i=2; i<N; i++) a[i]=1;
    for(i=2; i<N; i++) if(a[i]) for(j=i;(k=i*j)<N;j++) a[k]=0;
    for(i=2; i<N; i++) if(a[i]) printf("%d ",i); printf("\n");
    return 0;
}
```


```
> Smpl_num.exe 1000
```

Связный список

- Набор элементов, образующих «узел» (node), среди которых есть элемент «ссылка» (link).

  struct node { Item item; node *next; } ;

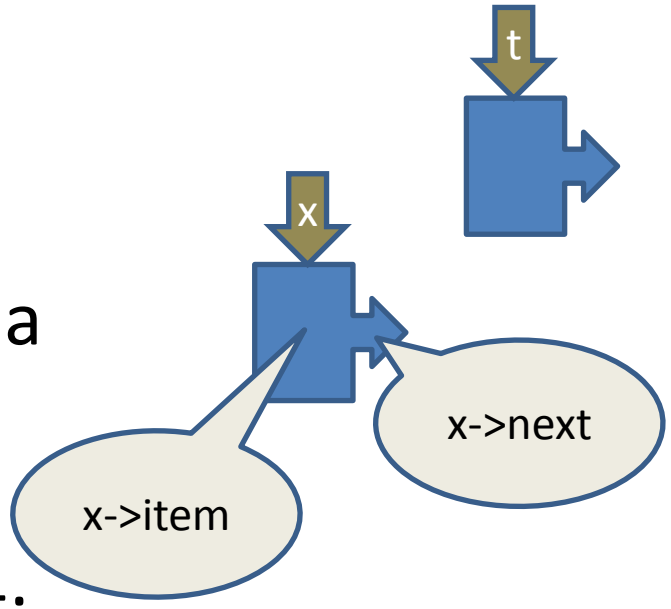
  typedef node *link;
- Создание узла

  link x = malloc(sizeof(node));
- Инициализация элементов узла

  x->item = …; x->next = …;
- Вставка узла t за узлом x

  t->next = x->next; x->next = t;
- Корректное удаление узла, следующего за x

  t = x->next; x->next=t->next; free(t);



Задача Иосифа

```
#include <stdio.h>
struct node { int item; struct node* next; };
typedef struct node* link;
link add(int x, link t) {
link a = malloc(sizeof(struct node));
    a->item = x; a->next = t; return a; }
int main(int argc, char* argv[]) {
int i, N=atoi(argv[1]), M=atoi(argv[2]);
link t = add(1,0);
link x = t;
    for(i=2; i<=N; i++) x=(x->next = add(i,t));
    while(N>1) {
        for(i=0; i<(M-1)%N; i++) x = x->next; N--;
        x->next = x->next->next; }
    printf("%d\n",x->item); }
```

Строки

```
#include <stdio.h>
#include <string.h>
int main() {
char a[100]={'H','e','l','l','o',',',' ','w','o','r','l','d','!','\0'};
char *b="Hello, world!";
    printf("sizeof(a)=%d strlen(a)=%d strlen(b)=%d\n",
                sizeof(a),strlen(a),strlen(b));
    return 0;
}
```

Элементарные операции со строками

- Вычисление длины строки (strlen(a))

```
char *b=a; while(*b++); return b-a-1;
```

- Копирование (strcpy(a,b))

```
while(*a++ = *b++);
```

- Сравнение (strcmp(a,b))

```
while(*a++ == *b++) if(!*(a-1)) return 0;
return *(a-1) - *(b-1);
```

- Конкатенация (strcat(a,b))

```
char a[N]={…,'\0'},*p=a;
while(*p)p++; while(*p++=*b++);
```

Потеря быстродействия

```
#include <stdio.h>
#include <string.h>
unsigned int start, end;
unsigned int access_counter() { asm("rdtsc"); }
int main(int argc, char *argv[]) {
int i,l1,l2;
    start=access_counter();
    l1=strlen(argv[1]); l2=strlen(argv[2]);
      for(i=0; i<l1; i++)
            if(!strncmp(argv[1]+i,argv[2],l2))
                  printf("%s %s %d\n",argv[1],argv[2],i);
    end=access_counter();
    printf("%u\n",end-start); }
```

Дополнительные способы группировки данных

Объединения

- способ наложения полей базового типа друг на друга:

```
union flasint { float a;
                char c[4]; };
```

Наборы битовых полей

- способ организации адресации объемов памяти отличных от байта:

```
struct structoffloat {
        unsigned mnts:23;
        int por:8;
        unsigned zn:1; };
```

```
#include <stdio.h>
struct structoffloat { unsigned mnts:23;
                       int por:8;
                       unsigned zn:1; };
union flt { float a;
            struct structoffloat b; } x;
int main() {
   for(x.a=.125f;x.a<16.f;x.a*=2.f)
   printf("%g\t%u %i %o\n",x.a,x.b.zn,x.b.por,x.b.mnts);
   for(x.a=-1.2f;x.a<1.3f;x.a+=.3f)
   printf("%g\t%u %i %o\n",x.a,x.b.zn,x.b.por,x.b.mnts);
   return 0; }
```

«Что позволено Юпитеру …»

```
#include <stdio.h>
#define N …
int main() {
int a[N],b[N];
   …
   a=b;          // ошибка
   …
return 0; }
```

```
#include <stdio.h>
#define N …
struct { int a[N]; } a,b;
int main() {
   …
   a=b;          //разрешено
   …
return 0; }
```

Составные структуры данных

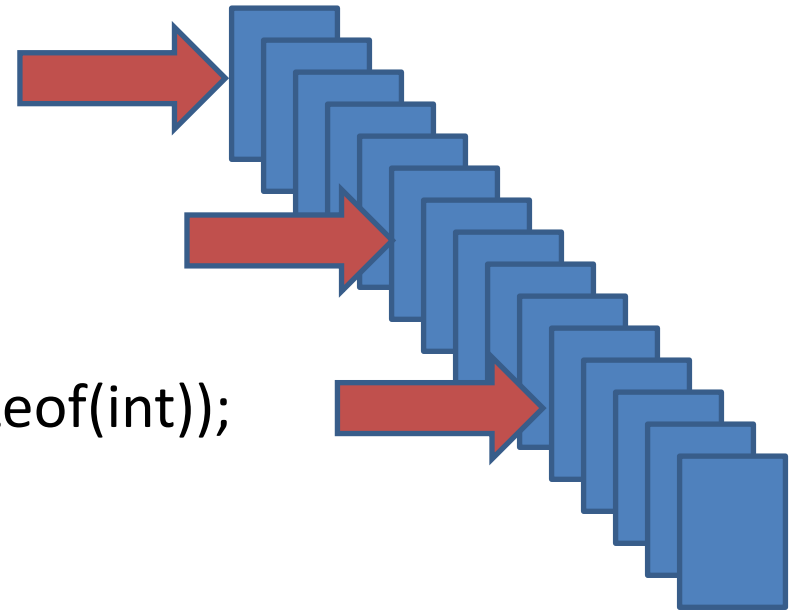
- Двумерные массивы

```
#include <iostream>
using namespace std;
int main(int argc, char** argv) {
int i, a[][3] = { {1,2}, {3,4,5}, {6,7} };
    cout << a[0][0] << ' ' << a[0][1]<< ' ' << a[0][2] << '\n';
    cout << a[1][0] << ' ' << a[1][1]<< ' ' << a[1][2] << '\n';
    cout << a[2][0] << ' ' << a[2][1]<< ' ' << a[2][2] << '\n';
    for(i=0; i<argc; i++} cout << argv[i] << '\t';
}
```

NB: при организации многомерного массива возможно не задавать только его первое измерение

Выделение памяти под двумерный массив

```
#include <iostream>
using namespace std;
const int N=3,M=5;

int main() { int i,j;
Int *a= (int*)malloc(N*M*sizeof(int));
     for(i=0;i<N*M;i++) a[i]=i;

int  **b = (int**)malloc(N*sizeof(int*));
     for(i=0;i<N;i++) b[i]=a+i*M;

  for(i=0;i<N;i++) for(j=0;j<M;j++) cout << b[i][j] << (j<M-1?'\t':'\n');
}
```



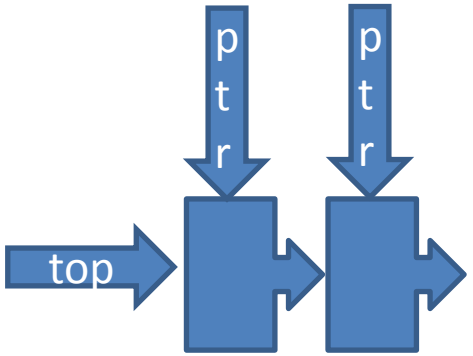
Массив массивов

```
#include <iostream>
using namespace std;
const int N=3,M=5;

int main() { int i, j, *a;
int** b = (int**)malloc(N*sizeof(int*));
    for(i=0; i<N; i++) b[i]=(int*)malloc(M*sizeof(int));
    a=b[0];

    for(i=0; i<N; i++) for(j=0;j<M;j++) b[i][j] = j+i*M;

    for(i=0; i<N*M; i++) cout << a[i] << ' ';
}
```

Исполнитель алгорифмов Маркова

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

struct P {
	string s1,s2;
	char b;
	P* next;
	P(void) { next=0; }
	friend istream& operator >>(istream& a, P &b)
			{ a >> b.s1 >> b.b >> b.s2;
			  if(b.s2=="*") b.s2=""; return a;}
};
```

```
int main(int argc, char** argv) {
   string word(argv[1]);
   P *top,*ptr;     ptr = top = new P;
   while( cin >> *ptr) { ptr->next = new P; ptr=ptr->next; }
   ptr = top;
   while(ptr->next) { int i=word.find(ptr->s1);
                if(i == string::npos) ptr=ptr->next;
                else { string bword(&word[0],&word[i]);
 string eword(&word[i+ptr->s1.size()],&word[word.size()]);
                     word = bword + ptr->s2 + eword;
                     if(ptr->b == '.') break; else ptr = top; }
                 }
   cout << word << endl; }
```



Текстовый препроцессор

- Расширение исходного кода
  #include <имя_файла>
  #include ”имя_файла”
- Вставить определение

NB!

```
#define sqr(x) x*x
A=sqr(a+b); // A=a+b*a+b;
```

- Определить константу
  #define имя константа
  #define array_size 128
- Условной компиляции

```
#if     #ifdef      #ifndef
        …
#else   #elif
        …
#endif
```

Текстовый препроцессор

- Операция # - преобразование лексемы в строку

```
#include <stdio.h>

#define Hello(x) printf("Hello, " #x "!\n")

int main() {
    Hello(world);
    return 0;
}
                                    // printf(“Hello, world!\n”);
```

Текстовый препроцессор

- Операция ## - конкатенации лексем

```
#include <iostream>
#include <conio.h>

#define Cat(x,y) x ## y

int main() {
    std::cout << "Input first char > ";
    Cat(get,ch)();
    std::cout << "\nInput second char > ";
    Cat(get,che)();
    return 0; }
```

Адсес	Имя переменной	Значение
22FF40	ptr	22FF44
22FF44	N	48
22FF48		
22FF4C		
22FF50	argc	
22FF54	argv	
4012D0	f(){…}	
401306	main(){…}	
4022A0	printf(){…}	
404000	s	PTR=&ptr=%p, ptr=&N=%p N=*ptr=**PTR=%d\n
404028	s1	&main()=%p &f()=%p &pririti()=%p\n
4050A0	PTR	22FF40

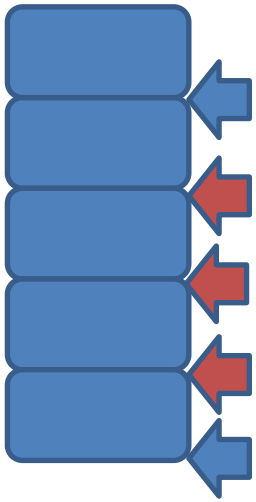
Сегмент стека

Сегмент программного кода

Сегмент инициализированных данных

Сегмент неинициализированных данных

Функции с неопределенным числом аргументов



Файл stdio.h

int printf(const char*,…);

int scanf(const char*,…);

```
#include <iostream>
#include <cstdarg>
int average( int first, ... );
int main() {
      std::cout << average( 2, 3, 4, -1 ) <<‘ ’;
      std::cout << average( 5, 7, 9, 11, -1 ) << ‘ ’;
      std::cout << average( -1 ) << ‘\n’;        }
int average( int first, ... ) {  int cnt = 0, sum = 0, i = first;
 void* mkr; mkr=&first + sizeof(first);
while( i != -1 ) { sum += i; cnt++;i =  *((int*)mkr)++;     }
   mkr=0;          return( sum ? (sum / cnt) : 0 ); }
```

Результат: 3 8 0

Операторы new и delete

- Пример

```
#include <stdio.h>
#include <malloc.h>
#define new(x) malloc(sizeof(x));
#define delete(x) free(x);
int main() {
    int i, *a, *ptr;
    ptr=a=new(int[10]);
    for(i=0;i<10;i++) *(ptr+i)=i;
    delete(a); }
```