

# The C programming Language

```
main() { printf(&unix["\021%six\012\0"], (unix)["have"] + "fun" - 0x60); }
```

## Part 2 of 3

- pointer type, referencing objects, untyped pointer, pass by value
- aggregate types
  - array
  - structure
  - union
  - enum
  - [], &, sizeof, ., ->
- bit fields
- data alignment, data structure padding
- namespaces: tags' names, members, labels, others
- type casts, operator ()
- usual array and function conversions
- type qualifiers and specifiers (volatile const signed)
- automatic, static, and extern storage classes
- dynamic memory allocation

## Pointers, pass by value

```
int x;  
int *xp;  
int **xpp;  
void *vp;
```

```
int a(void);  
int b();  
void c(void);
```

```
int d(int a, int b);  
int e(int *a, int *b);
```

```
int f(a, b)  
    int a;  
    char b;  
{ }
```

d(4,3);

## Aggregates

```
int ar0[]    = { 1, 2 };  
int ar1[2]   = { 1, 2 };  
int ar2[3]   = { 1, 2 };
```

```
{ struct s { int a; char b; };  
  union u  { int a; char b; };  
  enum e   { e0, e1 };
```

```
{ struct { char a, b; } ss = { 23, 32 };  
  union { long a, char b, int c } uu = { 2 };  
  enum ee { ee0 = 12, ee1, ee2 };
```

```
enum ee d = 24;
```

## Arrays vs pointers

```
int ar[] = { 1, 2, 3, 4 };  
ar[3] = 18;  
sizeof ar == sizeof (int [4]);  
ar[5] = 2;
```

```
int *p = ar;
```

```
p[2] == ar[2] ? 1 : 0;
```

```
void f(int ar[]);
```

*void f(int \*ar);*

Array and pointer: are they interchangeable?

When is array still an array?

*sizeof, &, init*

Is an array variable lvalue?

~~ar[0] = ar[1];~~

*non-modifiable  
lvalue*

## Multidimensional array and arrays' pointers

*no*

```
int ar2[3][2] = {{ 1, 2 }, { 3, 4 }, { 5, 6 }};  
int ar3[3][2] = { 1, 2, 3, 4, 5, 6 };  
ar2[1][0] == ar3[1][0];
```

*(*

```
int ar[] = { 1, 2, 3, 4 };  
ar[2] == *(ar + 2);
```

*address*

*1* int \*a[5];

*2* int (\*b)[5];

*3* b = &ar;

*4* int (\*\*c)[5];

c = &b;

## Accessing structure members

```
struct s {  
    int a, b;  
};
```

1 struct s s;  
2 struct s \*sp;

```
s.a = 4;  
s.b = 0xa;
```

3 sp = &s; &

```
sp->b == 10 ?
```

## Bit fields

```
struct t {  
    unsigned int a,  
    b : 2;  
} t;
```

— 4 bytes

— 2 bytes

t.a = 12;

t.b = 3;

t.b = 5; // 101

unsigned \*p;

1 p = &t.a;

2 ~~p = &t.b;~~



## Alignment and padding

```
struct a {  
    char  c0;  
    short s;  
    int   i;  
    char  c1;  
} a;
```

"self-aligned types"

c0	s	s	i
i	i	i	c1

c0		s	s
i	i	i	i
c1			

## Namespaces

```
struct a {  
    struct a *a;  
};
```

tags

fields

```
struct a (*a)(struct a (*a) {  
a:  
    return a;  
}
```

params

other

labels

## Type casts

```
int i = (int)2.8;
```

64

1 unsigned long long llx = 1 << 40;

2 unsigned long long llx = (unsigned long long)1 << 40;

```
struct s {  
    int i;  
    double d;  
};
```

```
struct s *sp;
```

```
int *ip = (int *)sp;
```

32  
int

int(32)

npnnpnTer

## Type qualifiers and specifiers

unsigned, signed

const

volatile

register

restrict

static

extern

auto

inline

## Storage classes

```
int a;  
static int b;  
extern int c;
```

```
static void f(void) {  
    auto int d;  
}
```

```
[extern] void g(void) {  
    ;  
}
```

$\text{int } f() = f()$

$d; \text{ auto } d;$

## Dynamic memory

```
#include <stdlib.h>
```

```
void *malloc(size_t size);
```

```
void free(void *ptr);
```

```
void *calloc(size_t nmemb, size_t size);
```

```
void *realloc(void *ptr, size_t size);
```

```
{
```

```
    int *p = malloc(sizeof *p);
```

```
    *p = 5;
```

```
    free(p);
```

```
}
```

# Home tasks

1. Write a function that finds the biggest element in an array of ints.
2. Create a linked list library with functions `list_add()`, `list_contains()`, `list_remove()`. Elements in a list should be dynamically allocated.
3. Create an integer array library with functions `ar_push()`, `ar_find_first()`, `ar_remove()`. Array should be allocated dynamically. The `ar_push()` function adds a new element after the last one; `ar_find_first()` returns the positive index of the first occurrence of the given element in the array or -1 if the number is not in the array; `ar_remove()` removes the element by its index. Order of elements in an array is not important.