Latihan Soal Orkom

Q: Array

Consider the following declarations:

```
short          S[7];
short         *T[3];
short        **U[6];
long double    V[8];
long double   *W[4];
```

Fill in the following table describing the element size, the total size, and the address of element i for each of these arrays.

Array	Element size	Total size	Start address	Element i
S	_____	_____	x_S	_____
T	_____	_____	x_T	_____
U	_____	_____	x_U	_____
V	_____	_____	x_V	_____
W	_____	_____	x_W	_____

A: Array

Observe that a pointer of any kind is 4 bytes long.
For IA32, gcc allocates 12 bytes for data type long double, even though the actual format requires only 10 bytes.

Array	Element size	Total size	Start address	Element i
S	2	14	x_S	$x_S + 2i$
T	4	12	x_T	$x_T + 4i$
U	4	24	x_U	$x_U + 4i$
V	12	96	x_V	$x_V + 12i$
W	4	16	x_W	$x_W + 4i$

Q: Matrix

Source code C:

```
int mat1[M][N];
int mat2[N][M];

int sum_element(int i, int j) {
    return mat1[i][j] + mat2[j][i];
}
```

Determine the values of constants M and N based on the assembly code!

Assembly code:

(*i* at %ebp*+8*, *j* at %ebp*+12*)

```
movl    8(%ebp), %ecx
movl    12(%ebp), %edx
leal    0(,%ecx,8), %eax
subl    %ecx, %eax
addl    %edx, %eax
leal    (%edx,%edx,4), %edx
addl    %ecx, %edx
movl    mat1(,%eax,4), %eax
addl    mat2(,%edx,4), %eax
```

A: Matrix

```
1       movl    8(%ebp), %ecx               Get i
2       movl    12(%ebp), %edx              Get j
3       leal    0(,%ecx,8), %eax            8*i
4       subl    %ecx, %eax                  8*i-i = 7*i
5       addl    %edx, %eax                  7*i+j
6       leal    (%edx,%edx,4), %edx         5*j
7       addl    %ecx, %edx                  5*j+i
8       movl    mat1(,%eax,4), %eax         mat1[7*i+j]
9       addl    mat2(,%edx,4), %eax         mat2[5*j+i]
```

We can see that the reference to matrix *mat1* is at byte offset 4(7i + j), while the reference to matrix *mat2* is at byte offset 4(5j + i). From this, we can determine that *mat1* has 7 columns, while *mat2* has 5, giving **M = 5 and N = 7**

Q: Structure (1)

structure declaration:

```
struct prob {
    int *p;
    struct {
        int x;
        int y;
    } s;
    struct prob *next;
};
```

What are the offsets (in bytes) of the following fields?

p: _____

s.x: _____

s.y: _____

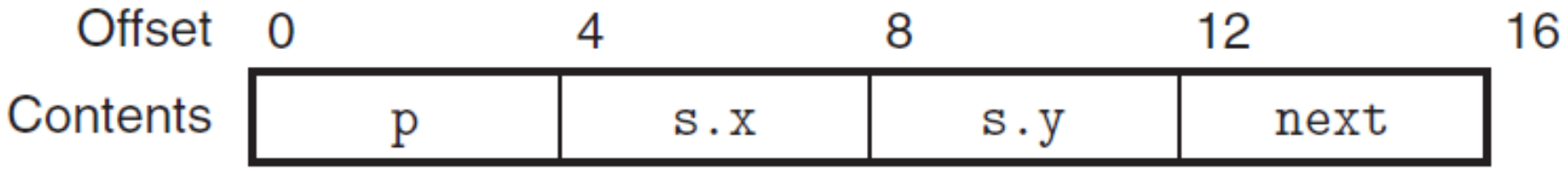
next: _____

How many total bytes does the structure require?

A: Structure (1)

This structure declaration shows that nested structures are allocated by embedding the inner structures within the outer ones.

The layout of the structure is as follows:



It uses 16 bytes.

Q: Structure (2)

The following procedure operates on the previous structure:

```
void sp_init(struct prob *sp)
{
    sp->s.x    = _____;
    sp->p      = _____;
    sp->next   = _____;
}
```

From this information, fill in the missing expressions in the code for *sp_init*

Assembly code of *sp_init* function:

```
sp at %ebp+8
    movl    8(%ebp), %eax
    movl    8(%eax), %edx
    movl    %edx, 4(%eax)
    leal    4(%eax), %edx
    movl    %edx, (%eax)
    movl    %eax, 12(%eax)
```

A: Structure (2)

```
    sp at %ebp+8
1       movl    8(%ebp), %eax     Get sp
2       movl    8(%eax), %edx     Get sp->s.y
3       movl    %edx, 4(%eax)     Store in sp->s.x
4       leal    4(%eax), %edx     Compute &(sp->s.x)
5       movl    %edx, (%eax)      Store in sp->p
6       movl    %eax, 12(%eax)    Store sp in sp->next
```

From this, we can generate C code as follows:

```
void sp_init(struct prob *sp)
{
    sp->s.x   = sp->s.y;
    sp->p     = &(sp->s.x);
    sp->next  = sp;
}
```

Q: Structure (3)

structure declaration:

```
struct {
    char      *a;
    short      b;
    double     c;
    char       d;
    float      e;
    char       f;
    long long  g;
    void      *h;
} foo;
```

Suppose it was compiled on a Windows machine, where each primitive data type of K bytes must have an offset that is a multiple of K.

A. What are the byte offsets of all the fields in the structure?

B. What is the total size of the structure?

C. Rearrange the fields of the structure to minimize wasted space, and then show the byte offsets and total size for the rearranged structure.

A: Structure (3)

Object size and by offset (original, 48 bytes):

Field	a	b	c	d	e	f	g	h
Size	4	2	8	1	4	1	8	4
Offset	0	4	8	16	20	24	32	40

The structure is a total of 48 bytes long. The end of the structure must be padded by 4 bytes to satisfy the 8-byte alignment requirement.

Object size and by offset (optimised, 32 bytes):

Field	c	g	e	a	h	b	d	f
Size	8	8	4	4	4	2	1	1
Offset	0	8	16	20	24	28	30	31

Q: Union

```
typedef union {
    struct {
        short  v;
        short  d;
        int    s;
    } t1;
    struct {
        int a[2];
        char  *p;
    } t2;
} u_type;
```

You write a series of functions of the form

```
void get(u_type *up, TYPE *dest) {
    *dest =  EXPR;
}
```

EXPR	TYPE	Code
up->t1.s	int	movl 4(%eax), %eax movl %eax, (%edx)
up->t1.v	_____	_____
&up->t1.d	_____	_____
up->t2.a	_____	_____
up->t2.a[up->t1.s]	_____	_____
*up->t2.p	_____	_____

A: Union

C declaration	Intel data type	Assembly code suffix	Size (bytes)
`char`	Byte	`b`	1
`short`	Word	`w`	2
`int`	Double word	`l`	4
`long int`	Double word	`l`	4
`long long int`	—	—	4
`char *`	Double word	`l`	4
`float`	Single precision	`s`	4
`double`	Double precision	`l`	8
`long double`	Extended precision	`t`	10/12

EXPR	TYPE	Code
`up->t1.s`	`int`	`movl 4(%eax), %eax` `movl %eax, (%edx)`
`up->t1.v`	`short`	`movw (%eax), %ax` `movw %ax, (%edx)`
`&up->t1.d`	`short *`	`leal 2(%eax), %eax` `movl %eax, (%edx)`
`up->t2.a`	`int *`	`movl %eax, (%edx)`

`up->t2.a[up->t1.s]`	`int`	`movl 4(%eax), %ecx`
		`movl (%eax,%ecx,4), %eax`
		`movl %eax, (%edx)`

`*up->t2.p`	`char`	`movl 8(%eax), %eax`
		`movb (%eax), %al`
		`movb %al, (%edx)`

Pembahasan Tugas Cache

t	s	b
xx	x	xx

No	Alamat	Binary	Hit/Miss
1	0	00000	hit
2	3	00011	
3	31	11111	
4	26	11010	
5	21	10101	
6	25	~~10000~~11001	
7	2	00010	
8	0	00000	
9	13	01101	
10	20	10100	

	v	tag	block
set 0	1	00	M[0-3]

	v	tag	block
set 1			