Homework 5

Keizou Wang

April 10, 2025

1. Suppose we are compiling for a machine with 1-byte characters, 2-byte shorts, 4- byte integers, and 8-byte reals, and with alignment rules that require the address of every primitive data element to be a multiple of the element's size. Suppose further that the compiler is not permitted to reorder fields. How much space will be consumed by the following array? Explain.

```
A : array [0..9] of record
    s : short
    c : char
    t : short
    d : char
    r : real
    i : integer
```

Each record has a size of $2+1+2+1+8+4 = 18$ bytes. Each index must align to a multiple of 8 bytes to align to ablock, so each is 24 bytes. $24 \cdot 10 = 240$ bytes.

2. For the following code specify which of the variables a,b,c,d are type equivalent.

```
Type T = array [1..10] of integer
     S = T
a : T
b : T
c : S
d : array [1..10] of integer
```

(a) Under structural equivalence.

```
a=b=c=d
```

(b) Under strict name equivalence.

```
a=b, c, d
```

(c) Under loose name equivalence.

```
a=b=c, d
```

3. We are trying to run the following C program:

```
typedef struct
{
    int a;
    char * b;
} Cell;

void AllocateCell (Cell * q)
{
    q = (Cell *) malloc ( sizeof(Cell) );
}

void main ()
{
    Cell * c;
    AllocateCell (c);
    c->a = 1;
    free(c);
}
```

(*a*) The program produces a run-time error. Why?

The programing produces a run-time error because the initialization and assignment to q on line 9 in the Allocate Cell function does not modify c. Therefore, c remains unintialized when the free function is called on it, causing the error.

(*b*) Rewrite the functions `AllocateCell` and `main` so that the program runs correctly.

```
typedef struct{
    int a;
    char * b;
} Cell;

void AllocateCell (Cell*& q){
    q = (Cell *) malloc ( sizeof(Cell) );
}

void main (){
    Cell* c;
    AllocateCell (c);
    c->a = 1;
    free(c);
}
```

4. Consider the following C declaration, compiled on a 32-bit Pentium machine.

```
struct
{
    int n;
    char c;
} A[10][10];
```

If the address of `A[0][0]` is 1000 (decimal), what is the address of `A[3][7]`? Explain how this is computed.

Size(int)=4, Size(char)=1 $\Rightarrow$ Size(Elem)=5
Size(Elem)=5, Size(Index)=$4k$: $k \in \mathbb{N}$ $\Rightarrow$ Size(Index)=8
Columns=10, x=3, y=7 $\Rightarrow$ Index=x(Columns)+y=37
Address = 1000 + Index $\cdot$ Size(Index) = 1296

5. Write a small fragment of code that shows how unions can be used in C to interpret the bits of a value of one type as if they represented a value of some other type (non-converting type cast).

```
union Number{
    int i;
    float f;
};

int main(){
    int original = 37;
    Number originalUnion = { .i=original };
    float unionInterpretedFloat = originalUnion.f; // 5.1848e-44
}
```