Programming Languages: Homework #5

Due on April 11, 2019 at 9:00am

Erin Keith Section 101

Michael DesRoches

Problem 1

(25pts) 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 an even multiple of the elements 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
```

Solution

A total of space consumed will be 240 bytes.
s has an offset of 0.
c has an offset of 2
t holds an offset of 4, compelled by 1 (means of allignment)
d holds an offset of 6
r holds an offset of 8
added up, we get 20 bytes - not a multiple of 8 so each array element is padded to 24 bytes
when adding, "i," which holds an offset of 16,
The array has 10 elements, 24 times 10 is equal to 240 bytes

Problem 2

(25pts) For the following code specify which of the variables a,b,c,d are type equivalent under (a) structural equivalence, (b) strict name equivalence, and (c) loose name equivalence.

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

Solution
(a) All have structural equivalence
(b) Variables a and b have strict name equivalence
(c) Variables a, b, and c have loose name equivalence

Problem 3

(25pts) 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);
}
```

The program produces a run-time error. Why? Rewrite the functions AllocateCell and mainso that the program runs correctly.

Solution

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

void AllocateCell ( Cell ** q )
{
  * q = ( Cell *) malloc ( sizeof ( Cell ));
}
void main ()
{
  Cell * c ;
  AllocateCell (& c ); // if you pass c then it is passed by value ,
                       // we use &c because we want to pass by reference
  c -> a =1;
  free ( c );
}
```

Problem 4

(25 pts) Consider the following C declaration, compiled on a 32−bit Pentium machine (with array elements aligned at addresses multiple of 4 bytes).

```
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.

Solution

```
struct
{
int n ;
char c ;
} A [10] [10]; // row , colomn total size of 10
```

The address of A[3][7] is 1296 So the size of the structure is 8. The base address of A[0][0] is 1000. Now we do the computations:
(1000) + (3 10 8) + (7 8) = 1296

COMPUTATION:
(base address) + (element row 3 * total size of rows * size of struct) + (element column 7 * size of struct)

Problem 5

(Extra Credit −10 pts) 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).

Solution

```
union ch_ascii
{
    char ch ;
    unsigned int asc_val ;
} union_tag ; // union variable

void main ()
{
    union_tag . ch =    C   ;
    union_tag . asc_val = (int )( union_var . ch ); // non converting type
                                                    //cast occuring here
    // output of program C and 67
    printf ("% c  %d", union_tag . ch , union_tag . asc_val );
    return 0;
}
```