C Union (Answers)

Sayak Haldar
IIEST, Shibpur

1) Answer is c) Biggest member in the union

2) Answer is a) a

Explanation: union temp s = {1,2.5,’A'};
Here, {} works as an operator and choose the leftmost value to write in the allocated memory for union.

As a result,

```
#include <stdio.h>
union temp
{
        char c;
        float b;
        int a;

};
int main()
{
        union temp s={'A',2.5,1};
        printf("%f %d %c\n",s.b,s.a,s.c);
        return 0;
}
```

This will print 0.000000 65 A

```
#include <stdio.h>
union temp
{
        float b;
        int a;
        char c;

};
int main()
{
        union temp s={2.5,1,'A'};
        printf("%f %d %c\n",s.b,s.a,s.c);
        return 0;
}
```

This will print 2.500000 1075838976

And

```
#include <stdio.h>
union temp
```

```
{
        int a;
        float b;
        char c;

};
int main()
{
        union temp s={1,2.5,'A'};
        printf("%f %d %c\n",s.b,s.a,s.c);
        return 0;
}
```

this will print 0.000000 1 #

3) Answer is c) 40

Explanation: size of union=Biggest member in the union. Here, biggest member of the union is the array b of type int

5) Answer is a) will be large enough to hold the largest of the three types

6) Answer is c) Both a & b

7) Answer is d) Both a & b

Explanation: I guess no explanation is needed for symtab[i].u.sval[0]

But, a little explanation is needed for *symtab[i].u.sval
*symtab[i].u.sval
. this is Element selection by reference operator
Whereas,
* this is Indirection (Reference) operator
Now, '.' operator has higher precedence than '*' operator
'.' operator has associativity from left to right and '*' operator has associativity from right to left

*symtab[i].u.sval
*(symtab[i].u.sval)
symtab[i[.u.sval[0] (Derived by pointer arithmatic)

8) Answer is c) 4

9) Answer is d) 5

10) Answer is d) sizeof(int)

Explanation:

Now, you must be confused with the way the union defined here. For defining a new union datatype , you need to use union statement. The union statement defines a new data type, with more than one

member for your program. The format of the union statement is as follows:

The format of the union statement is as follows:

```
union [union tag]
{
  member definition;
  member definition;
  ...
  member definition;
} [one or more union variables];
```

the fields mentioned under [] are optional
So, now check the program again.

```
1. union
2. {
3.         int x;
4.         char y;
5. }p;
```

This is how the union is defined. Is not it? Clearly, you can see, the optional union tag field is missing. This will cause any problem until we want to create variable of this newly defined union from other portions of the code. The problem with this kind of union is that the variables of the newly defined union must be created during the definition of the union(Like, variable p is created during union definition). But, in the program only p variable of the newly defined union is used. So, it will not cause compilation error. And size of union=size of the biggest member of the unionn (which is x of int type). So, size of p will be 4

11) Answer is c) Depends on the compiler

Explanation: In case of some compiler. The size allocated for the union=the biggest member of the union (Always). In case of of other compilers, the size allocated for the union=the member you are trying to make active explicitly. Like, in thiss case, the program is trying to make member 'y' of type type char active explicitly.

12) Answer is c) 60

Explanation: Union tag name can be later used as a variable name

13) Answer is d) 1

14) Answer is b) 97

15) Answer is b) Inplementation depended

Explanation: It is implementation depended.

For compilers like gcc (in ubuntu 14.04 which I am using)

The memory allocated for the union is 4 bytes and integer value 4 is stored in the memory location. Now, we are trrying to retrive the intger value 10 stored in the memory location as a float. So, it will be retrieved as 0.000000 (It is a compilecated thing. Since, float can hold value upto 10^{47} . So, reading a small integer value from a memory allocated for int datatype as a float or retrieve/fetch a ssmall integer value from a memory allocated for int datatype as a float will produce these kind of results.)

For, other compilers, maybe the result is 10.000000

(The question is, is there any concept of default active member in case of union in c, if two datatype of same size are present as the members with biggest size of the union)

More Explanation:

The union is only as big as necessary to hold its largest data member. The other data members are allocated in the same bytes as part of that largest member. The details of that allocation are implementation-defined, and it's undefined behavior to read from the member of the union that wasn't most recently written. Many compilers implement, as a non-standard language extension, the ability to read inactive members of a union.

So, here, the float variable and int variable (which are members of the union, share the same memory slot of 4 bytes which is allocated for the union. Since, the allocation is implementation depended, the result is implementation depended. (So, in these type of questions, if implementation depended is present as an option, most of the cases it is the right answer)

16) Answer is b) 1 & 2

17) Answer is c) 8

Explanation: The union is only as big as necessary to hold its largest data member. The other data members are allocated in the same bytes as part of that largest member.

18) Answer is b) 97 97

Explanation: The union is only as big as necessary to hold its largest data member. The other data members are allocated in the same bytes as part of that largest member. The details of that allocation are implementation-defined, and it's undefined behavior to read from the member of the union that wasn't most recently written.

However, in most of the cases, the value allocated to the biggest member is the value shown when we are trying the read other members, if other members are big enough to hold the value.

19) Answer is b) 4

References:

1) http://www.sanfoundry.com/c-interview-questions-answers/