innovate achieve lead

BITS Pilani

Lecture 14

Tagged union data types, sets and pointers data type

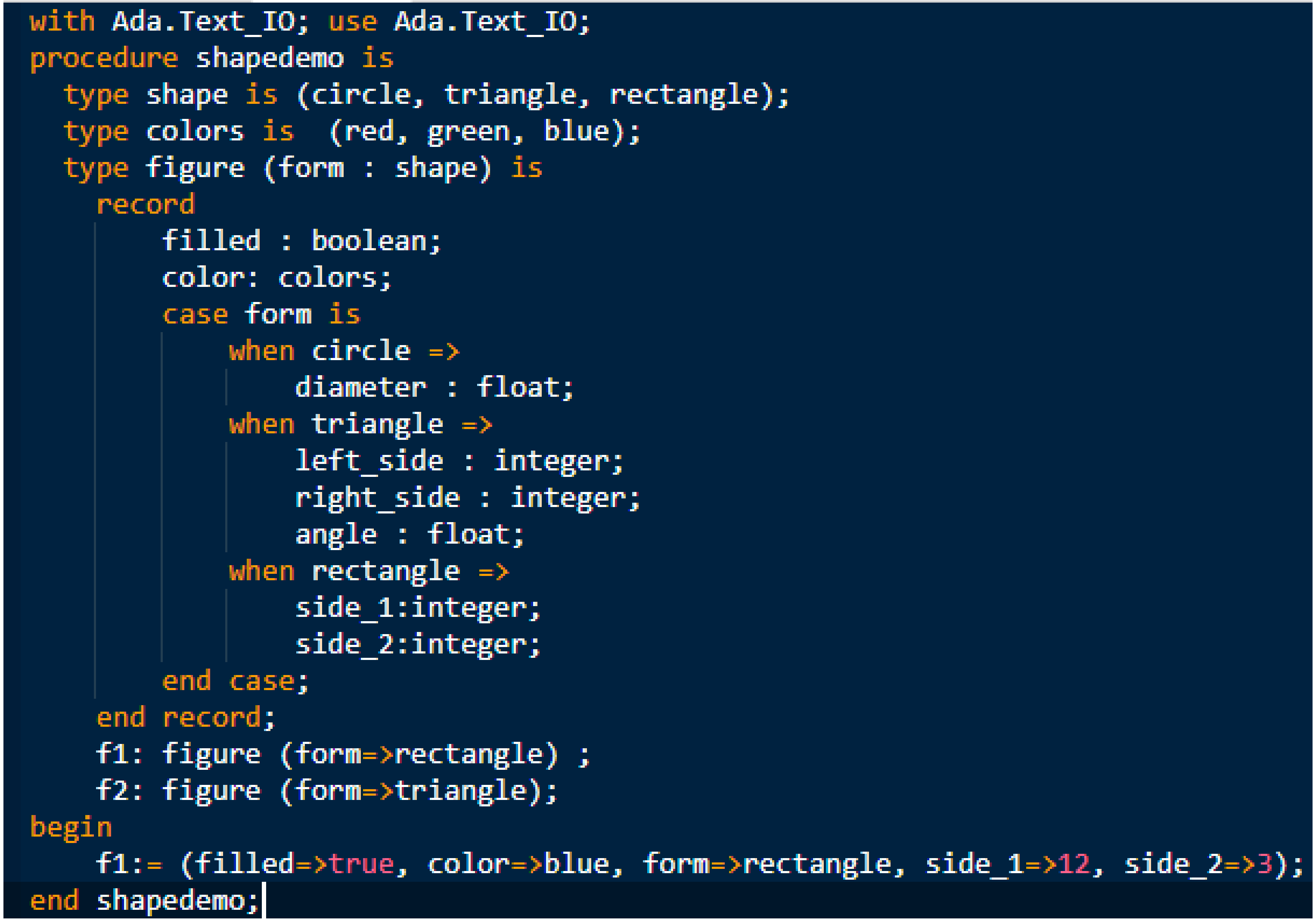
Union data type

- The variables of union data type may store values of different types at different times during execution of the program.
- C and C++ provide union constructs, but type checking is not in the design of the construct.
- This union construct is also known as **free union**.
- Type checking in union data type required an indicator for field usage.

Tagged or Discriminated union data type

- An indicator, also known as **tag** or **discriminator**, is used to specify the latest field usage for the union variables.
- ALGOL 68 was the first language to provide support for unions with tag.
- Tagged union is supported in ADA, ML, Haskell and F#.

Ada union type



```
with Ada.Text_IO; use Ada.Text_IO;
procedure shapedemo is
  type shape is (circle, triangle, rectangle);
  type colors is  (red, green, blue);
  type figure (form : shape) is
    record
        filled : boolean;
        color: colors;
        case form is
            when circle =>
                diameter : float;
            when triangle =>
                left_side : integer;
                right_side : integer;
                angle : float;
            when rectangle =>
                side_1:integer;
                side_2:integer;
        end case;
    end record;
    f1: figure (form=>rectangle) ;
    f2: figure (form=>triangle);
begin
    f1:= (filled=>true, color=>blue, form=>rectangle, side_1=>12, side_2=>3);
end shapedemo;
```

Fields of the record data type

- filled, color and form of different types each.
- Type of field ‘filled’ is Boolean
- Type of field ‘color’ is colors
- Type of ‘form’ is union which can be represented as follows (indicating use of one)

  <circle | triangle | rectangle>

[circle x triangle x rectangle may represent the structure]

- Hence, the type expression for the **figure** record is

boolean x triangle x <circle | triangle | rectangle>

Type expression of discriminated variable **f1** of **figure** data type

- Variable declaration

  f1: figure (form=>rectangle) ;

- Type expression of f1

  type expression of figure x tag information

boolean x triangle x <circle | triangle | rectangle> x (tag=rectangle)

- Field access by f1

  f1. diameter = f1.left_side+2;

- How to keep a check on correct access?

  Use type expression of f1 as above (and not that of the figure alone)

What can go wrong with f1 or f2?

- If the following is seen at compile time,

```
        f1. diameter = f.left_side+f1.right_side
```

The compiler knows

```
        f1:figure(form=>rectangle);
```

Then the type expression, includes information about the usage of the variable f1

Ada Union

- **Static type checking**: The variable f2 is declared constrained to be a triangle and cannot be changed to another variant.
- This is an example of discriminated union.
- This way the type checking is done at compile time.
- Therefore the possibility of any access to wrong data is prevented by reporting this as an error and making the data access type safe.

Ada union

- **Dynamic type checking:** The unconstrained variant record variable declared as

```
f1: figure;
```

The variable f1 has no initial value or a discriminator (tag). Therefore if the code has a initialization later in another time instance as shown below, then the type checking is done at run time.

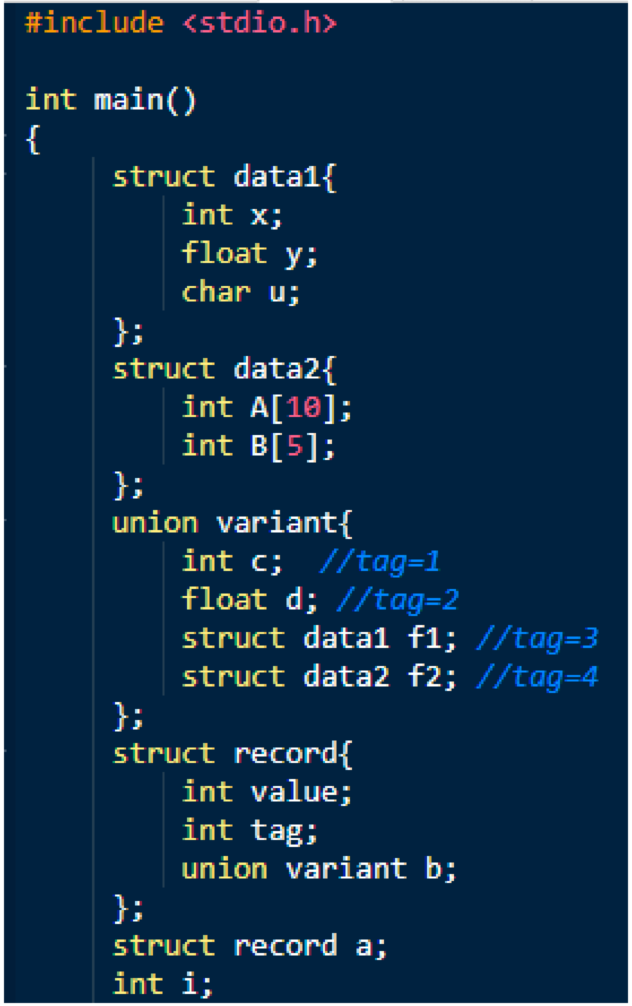
```
f1:= (filled=>true, color=>blue, form=>rectangle,
side_1=>12, side_2=>3);
```

Implementation of Union types

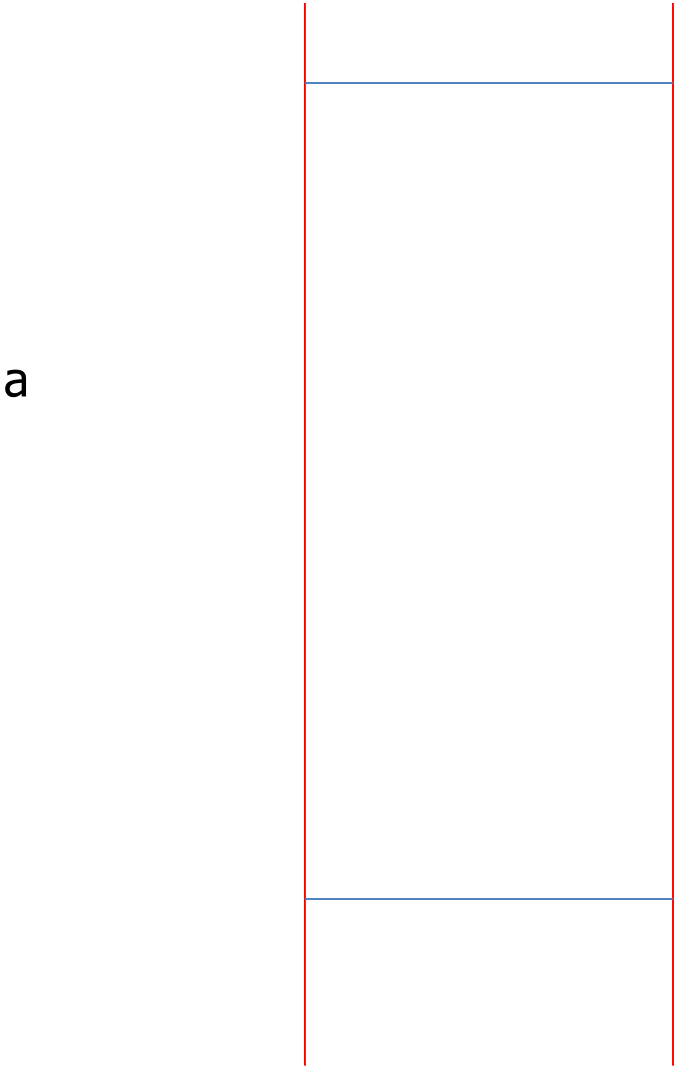
- The same address is used for all possible variants.
- Sufficient storage is allocated to the largest variant.
- The tag can be the part of the fixed part of the variant record. The tag can indicate use of the variant part of the record.

Tagged union example using C language

1. Type definition

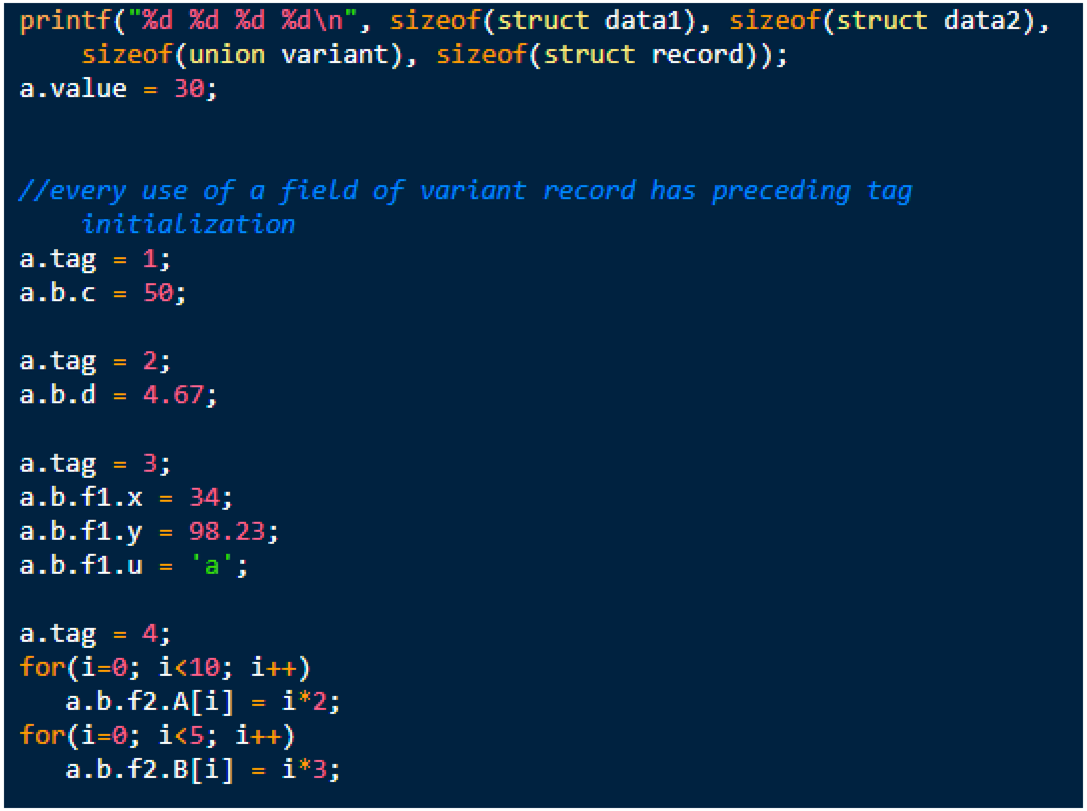


```
#include <stdio.h>

int main()
{
    struct data1{
        int x;
        float y;
        char u;
    };
    struct data2{
        int A[10];
        int B[5];
    };
    union variant{
        int c;  //tag=1
        float d; //tag=2
        struct data1 f1; //tag=3
        struct data2 f2; //tag=4
    };
    struct record{
        int value;
        int tag;
        union variant b;
    };
    struct record a;
    int i;
```



Tagged union example using C language

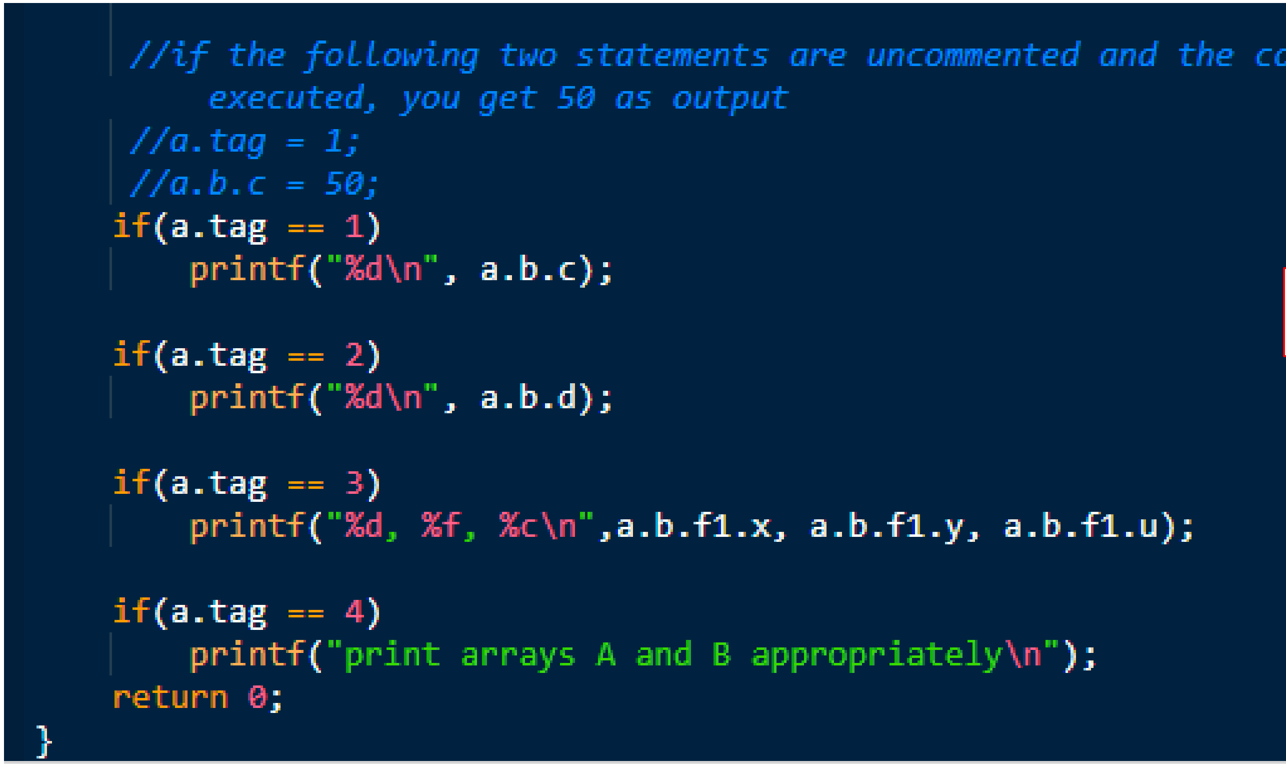
2. Use of tag with every new assignment to fields of variant record



```
printf("%d %d %d %d\n", sizeof(struct data1), sizeof(struct data2),
    sizeof(union variant), sizeof(struct record));
a.value = 30;


//every use of a field of variant record has preceding tag
    initialization
a.tag = 1;
a.b.c = 50;

a.tag = 2;
a.b.d = 4.67;

a.tag = 3;
a.b.f1.x = 34;
a.b.f1.y = 98.23;
a.b.f1.u = 'a';

a.tag = 4;
for(i=0; i<10; i++)
   a.b.f2.A[i] = i*2;
for(i=0; i<5; i++)
   a.b.f2.B[i] = i*3;
```

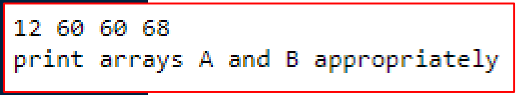
Tagged union example using C language

3. Use of tag with every field data access



```
    //if the following two statements are uncommented and the code is executed, you get 50 as output
    //a.tag = 1;
    //a.b.c = 50;
    if(a.tag == 1)
        printf("%d\n", a.b.c);

    if(a.tag == 2)
        printf("%d\n", a.b.d);

    if(a.tag == 3)
        printf("%d, %f, %c\n",a.b.f1.x, a.b.f1.y, a.b.f1.u);

    if(a.tag == 4)
        printf("print arrays A and B appropriately\n");
    return 0;
}
```

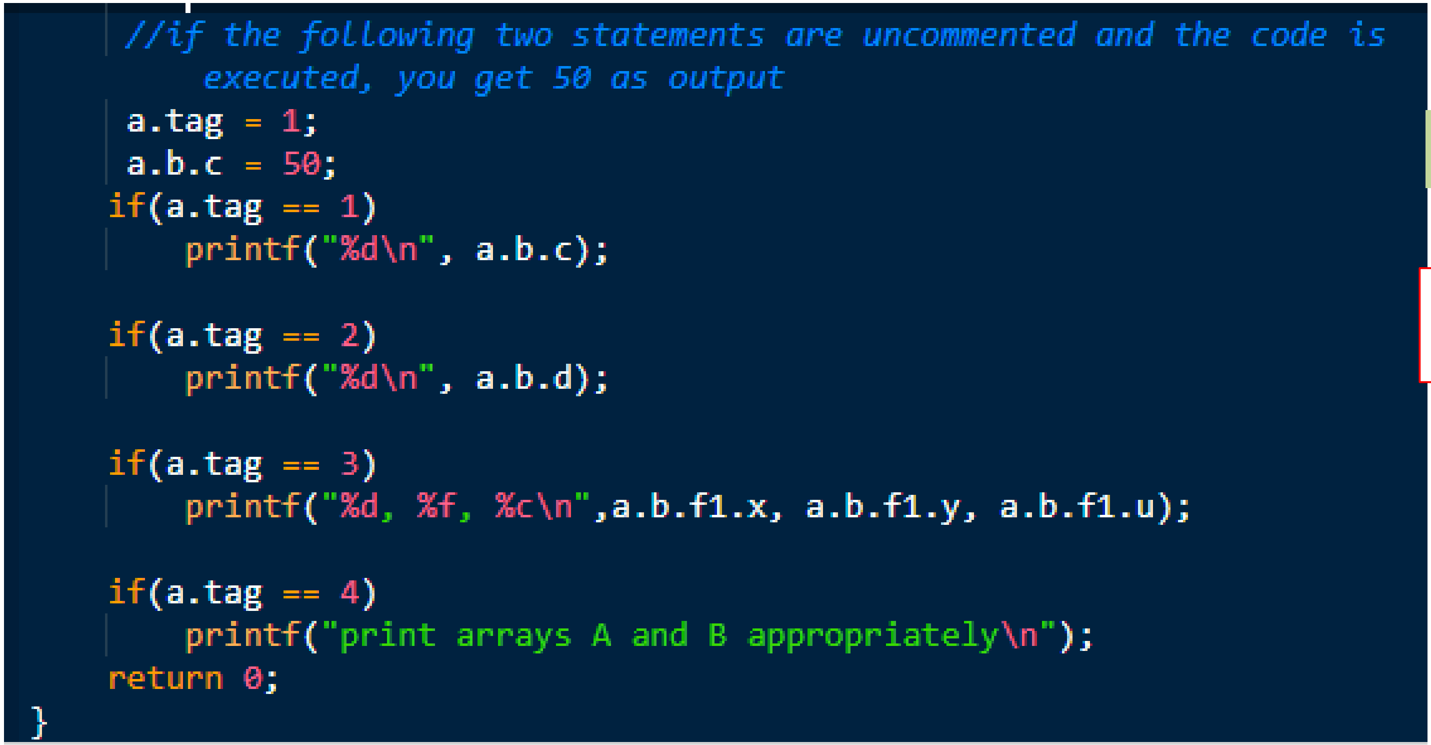
output



```
12 60 60 68
print arrays A and B appropriately
```

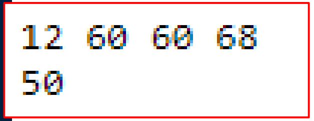
Tagged union example using C language

The latest value of tag is used for dynamic type checking.



```
     //if the following two statements are uncommented and the code is
         executed, you get 50 as output
     a.tag = 1;
     a.b.c = 50;
    if(a.tag == 1)
        printf("%d\n", a.b.c);

    if(a.tag == 2)
        printf("%d\n", a.b.d);

    if(a.tag == 3)
        printf("%d, %f, %c\n",a.b.f1.x, a.b.f1.y, a.b.f1.u);

    if(a.tag == 4)
        printf("print arrays A and B appropriately\n");
    return 0;
}
```

output



```
12 60 60 68
50
```

Sets

- A set of n numbers, each less than n, is implemented by a bit vector of n bits
- Example: set A={ 3, 6, 1, 8, 15, 0, 2}

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	0	0	0	1	0	1	0	0	1	1	1	1

- Represented- by a simple integer word.
- limitation - only small element values can find their place as a bit (else two or more words are used)

Sets

- Operations on sets are implemented as bit operations
- Example (in C) (not supported by C directly)
- ➢ AddElement(set A, element e):

```
        mask=1;
        A=A | (mask<<e); // use left shift operator
```

- ➢ set union: bitwise OR
- ➢ set intersection: bitwise AND
- ➢ set complement: bitwise NOT
- ➢ isMember(set A, element e): if ((1<<e)!=0) then e$\in$A

Sets (in Pascal)

- Pascal supports the set type.
- Syntax:

  var A: set of [1..3]

- The variable A can denote one of the following sets

  [], [1], [2], [3], [1,2], [2,3], [3,1], [1,2,3]

- The subset [1,3] can be denoted by the bit string 101