# Data Structures and Algorithms [17ECSC204]
# Other Important Concept Notes
**III-D 2018-19 SoCSE**

## Unions

Below given is a program, followed by properties. First go through the program, and then read the properties of Unions. Revisit the program again and understand the unions better.

**A program to demonstrate the Unions**

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

enum type
{
   INT, FLOAT, STRING
} tag;

union sym_value
{
  int ival;
  float fval;
  char *sval;
};

int get_input(union sym_value *u)
{
   int choice = 0;
   printf("Enter\n1- initilaize int\n2-initilaize float\n3-initilaize string\n");
   scanf("%d", &choice);
   switch(choice)
   {
      case 1: u->ival = 20;
            tag = INT;
            break;

      case 2: u->fval = 200.0;
            tag = FLOAT;
            break;
```

```
        case 3: u->sval = "string";
                tag = STRING;
                break;

    }
    return tag;
}

int main()
{
    union sym_value u1;
    union sym_value u2;
    int result = 0;

    printf("Size of u1 Variable is... %d\n", sizeof(u1));
    u1.ival = 10;
    u1.fval = 10.0;
    u1.sval = "abc";
    printf("Integer is..%d\n", u1.ival);
    printf("Float is..%f\n", u1.fval);
    printf("String is..%s\n", u1.sval);

    u2 = u1;
    printf("String of u2 variable is..%s\n", u2.sval);

    result =  get_input(&u2);
    printf("The data initialized in Function is..\n");
    if(result == 0)
         printf("Integer: %d\n", u2.ival);
    else if(result == 1)
         printf("Float : %f\n", u2.fval);
    else
        printf("String: %s\n", u2.sval);

    return 0;
}
```

**Note:** Refer Class Notes for a much simpler program.

**Characteristics of UNIONS:**

- The size of memory allocated by the compiler is the size of the member that occupies largest space
- Memory allocated is shared by individual members of union
- The address is same for all the members of a union
- Only one member can be accessed at a time
- Only the first member of a union can be initialized

## Bit Fields

A group of several bits can be packed together using a structure. A member of a structure that is composed only of a specified number of bits is called bit-fields.

**Syntax:**

```
struct tag_name
{
        data_type   member01:  bit_length;
        data_type   member02:  bit_length;
        …
        …
};
```

**Program to demonstrate the Bit Fields**

```
#include <stdio.h>
struct student
{
   char *name;
   unsigned sem : 3;
   unsigned courses_registered : 3;
   unsigned eligibility : 1;
};

int main()
{

   struct student s1;
   int sem, courses_registered, eligibility;

   s1.name = "aaa";
   printf("Enter the sem, courses registered and is student eligible data\n");
   scanf("%d%d%d",&sem,&courses_registered, &eligibility);

   s1.sem = sem;
   s1.courses_registered = courses_registered;
```

```
    s1.eligibility = eligibility;

    printf("The entered details are...\n");
    printf("Name: %s\nSem: %d\nCourses Registered: %d\nEligibility: %d\n", s1.name, s1.sem,
s1.courses_registered, s1.eligibility);

    return 0;
}
```

**Characteristics:**

- Addresses of bit fields cannot be accessed. So, we cannot read the values into the bit fields using scanf()
- Pointers cannot be used to access bit fields
- Bit fields cannot be declared as static
- Bit fields should be assigned the values within the range. Otherwise the behavior of the program will be unpredictable
- No field can be longer than 1 long word (32 bits)
- It is not possible to declare array of bit fields

## Infix to Prefix Conversion

Let us understand with an example. Input: A * B + C / D

**Step 1:** Reverse the given input
Reversing the given expression input we have:
D / C + B * A

Note: while reversing convert '(' to ')' and ')' to '('.

**Step 2:** Follow the process carried for postfix string conversion

| Symb | String | Opnd Stk |
|---|---|---|
| D | D | |
| / | D | / |
| C | DC | / |
| + | DC/ | + |
| B | DC/B | + |
| * | DC/B | +* |
| A | DC/BA | +* |
| | DC/BA*+ | |

**Step 3:** Reverse the output.
Final converted expression is: +*AB/CD

## Evaluating a Prefix Expression

Let us understand with an example.

Input: -*+4325

- Write the table by reading the symbols from reverse
- When we find an operator, the first operator popped will be opnd1
- The remaining procedure is same as postfix evaluation

| Symb | Opnd1 | Opnd2 | Value | Stack |
|---|---|---|---|---|
| 5 | | | | 5 |
| 2 | | | | 5, 2 |
| 3 | | | | 5, 2, 3 |
| 4 | | | | 5, 2, 3, 4 |
| + | 4 | 3 | 7 | 5, 2, 7 |
| * | 7 | 2 | 14 | 5, 14 |
| - | 14 | 5 | 9 | **9** |

9 is the result.

## Quadratic Probing

A quadratic probing uses the hash function of the form:

$$h(k, i) = (h'(k) + c_1 i + c_2 i^2) \bmod m$$

where h' is auxiliary hash function, c1 and c2 are not equal to 0 and are auxiliary constants, i = 0, 1, … m-1. The initial position probed is T[h'(k)]; later positions probed are offset by amounts that depend in quadratic manner on the probe i. This method works much better than linear probing.

Or in simpler terms, to avoid clustering effect, instead of incrementing the indexing variable by 1, we increment by $i^2$.

You can refer to other internet resources if you need to understand any better.

## Input and Output Restricted Queues

An input restricted queue is a double ended queue, which allows insertion at only 1 end, rear end, but allows deletion at both ends.

An output-restricted queue is a queue which allows deletion at only one end, front end, but allows insertion at both ends, rear and front ends.

## More Concepts on Hashing

### Open and Closed Hashing

Open Hashing (Separate Chaining):
In open hashing, keys are stored in linked lists attached to cells of a hash table. (Scatter table concept)

Closed Hashing (Open Addressing):
In closed hashing, all keys are stored in the hash table itself without the use of linked lists. (all other techniques)

### Division Method

In this method we map a key k into one of m slots by taking the remainder of k divided by m. the hash function is h(k) = k mod m.
For example: if the hash table has size m = 12 and key k = 100, then h(k) = 4. Since it requires only a single division operation, hashing by division is quite fast.

## File Error Functions

**clearerr()** → functions clears/resets all error indicators corresponding to the file pointed to by the file pointer passed as argument.
Syntax: void clearerr(FILE *fp)

**ferror()** → the function tells whether any previous operation on the file was successful or produced any kind of error.
Syntax: int ferror(FILE *fp)
ferror() must be called immediately after any operation on the file otherwise any error produced may be lost. Its returns non-zero if any error was produced and 0 otherwise.

**perror()** →this function prints a user defined character string and an implementation defined string corresponding to the global variable errno.
Syntax: void perror(char *str)