Introduction to Data Structure

Farah Zakiyah Rahmanti

2014

Definition

- Organization scheme, ex : structure and array.
- Applied to the data, so that the data can be interpreted and specific operations can be performed on the data.

Background

- Array data type likely to use a fixed size
- The size of an object can not be certained until the execution time.
- Providing memory allocation facility to make the buffer size and dynamic arrays.
- Dynamic memory means that the space will be allocated when the program is executed.

Pointer

- A variable is used to save address of data
- Contain of pointer isn’t a value of data, but address of data
- Dynamic memory allocation

Pointer

- Common shape :

```
int *data_address;
```

- Knowing address of data, can be used “&” :

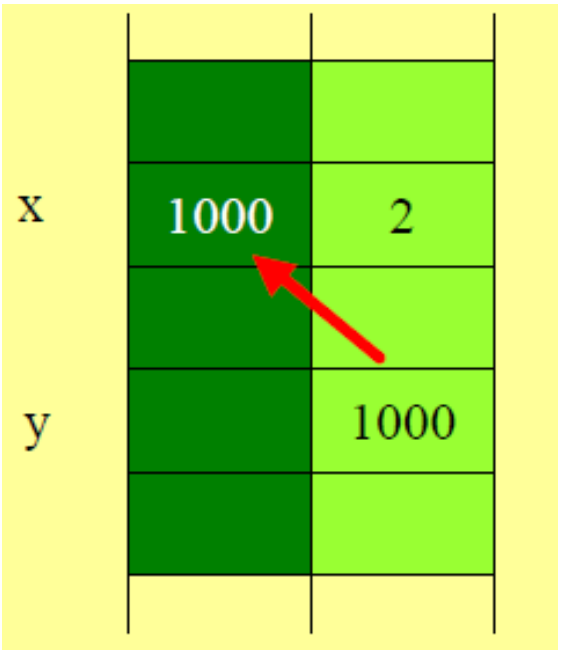
```
data_address = &data;
```

Pointer

```
int x;
int *y;


...


x = 2;
y = &x;
```



Sizeof ()

- Getting the size of data type, variable, structure
- Return value : size of the object (byte)
- Parameter : an object or a data type

```
typedef struct employee_st {
   char name[40];
   int id;
} employee;

main () {
   int myInt;
   employee john;

   printf("size of int is %d\n", sizeof(myInt));
   printf("size of int is %d\n", sizeof(int));
   printf("size of Employee is %d\n", sizeof(Employee));
   printf("size of john is %d\n", sizeof(john));
   printf("size of char is %d\n", sizeof(char));
   printf("size of short is %d\n", sizeof(short));
   printf("size of long is %d\n", sizeof(long));
   printf("size of float is %d\n", sizeof(float));
   printf("size of double is %d\n", sizeof(double));

   return 0;
}
```

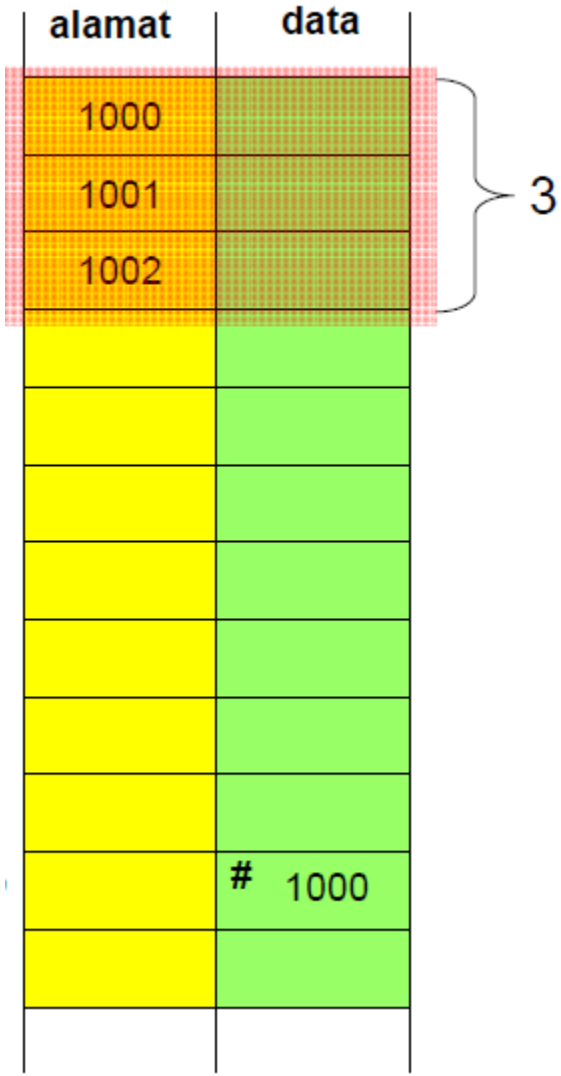
Malloc ()

- Standard function that's used to memory allocation
- Void *malloc (int num_byte)
- Return value : pointer to void

Malloc ()

```
int *x;
X = (int *)malloc(3*sizeof(int));
if (x==NULL) {
  printf("error on malloc\n");
  exit(0);
} else {
  // ...
}
```

```
int *p;
P = (int *)malloc(3*sizeof(int));
```



Free()

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

Main()
{
  char *pblok;
  pblok = (char *)malloc(500*sizeof(char));

  if (pblok == NULL)
     puts("error on malloc");
  else {
  puts("OK, memory allocation has been done");
  puts("--------");
  free(pblok);
  puts("free memory well done");
  }
}
```

Realloc ()

```
...
  pblok = (char *)malloc(500*sizeof(char));
...
...
  pblok = realloc(pblok, 600*sizeof(char));
```

Struct

- Struct is a way to classify data with the same name, but different types.

- Declaration of struct :

```
struct nameofstruct {
   type data1;
   ...
   type datan;
} var1, ..., varn;
```

Struct declaration

```
typedef struct dataof_student {
  int nrp;
  char name[20];
};
struct dataof_student student;

typedef struct dataof_date{
  int date;
  int month;
  int year;
} birthday;

Struct dataof_date birthday;
```

Struct initialization

```
struct dataof_student {
  int nrp;
  char name[20];
} student = {7, “agus”};
```

How to access struct

```
struct dataof_date{
  int date;
  int month;
  int year;
} birthday;

birthday.date  = 17;
birthday.month = 8;
birthday.year  = 1945;
```

Struct in struct

```
struct dataof_date{
   int date;
   int month;
   int year;
};

struct dataof_student {
   int nrp;
   char name[20];
   struct dataof_date birthday;
} student;

Student.birthday.date = 17;
```

Linear List

- Linear ordered list
- Notion of :
  - First element
  - Next element : successor, predecessor
  - Address

List Primitives

- CreateEmpty
- IsEmpty
- Insert, update, delete
- Concatenation
- Traversal
- Searching
- Find the length (number of element) of a list
- Find maximum/minimum value of list elements

List Physical Representation

- Contiguous by table/array
- Linked representation
  - Pointer
  - Table

List Variation

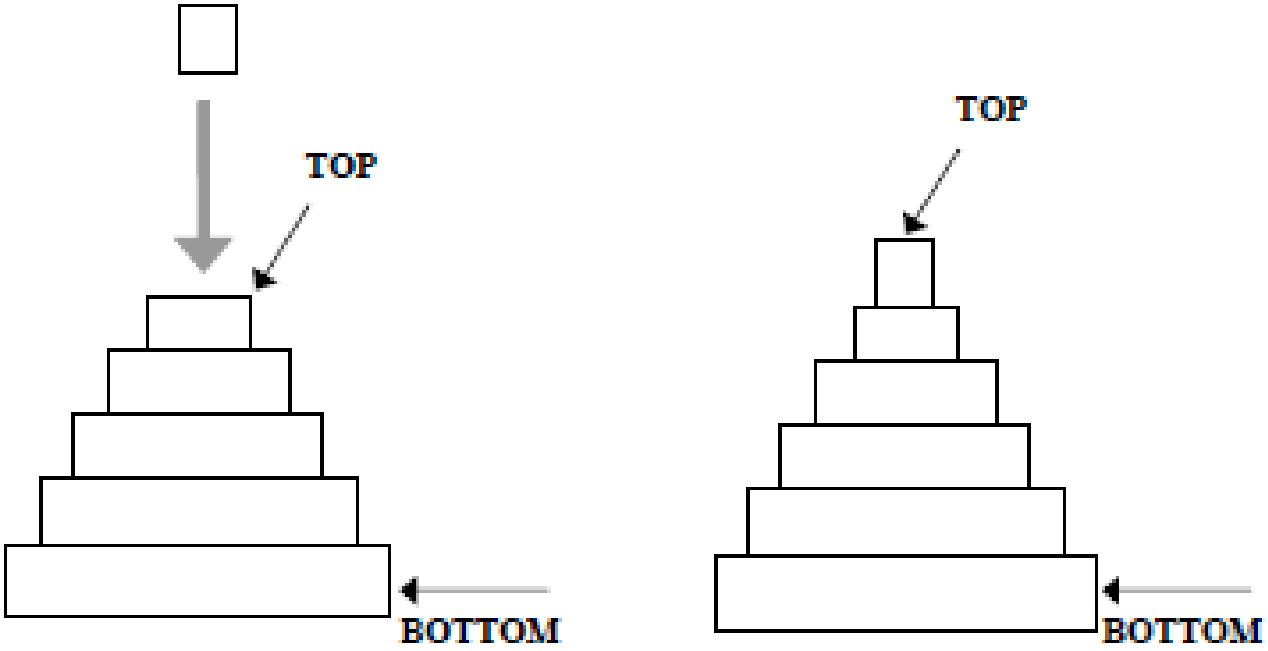
- List with first and last element
- Circular list
- Double pointer list
- List with dummy element
- Possible combinations

Stack (1)

- Ordered list in which all insertions or deletions are made at one end, called the TOP
- Last in First Out (LIFO)
- Application in computer science :
  - Dynamic memory management
  - Arithmetic calculation
  - Recursive subprogram call
  - Backtracking algorithms

Stack (2)

- Notion of top, element, empty



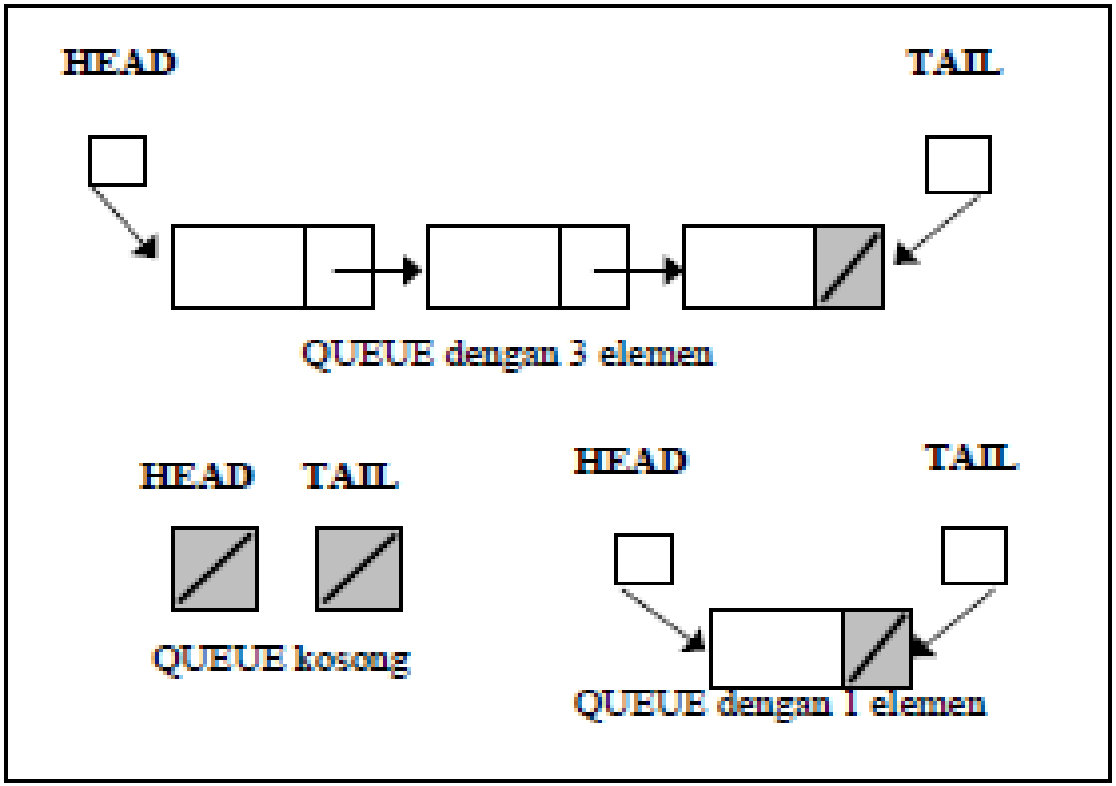
Queue (1)

- Ordered list in which all insertions take place at one end (the rear/tail), while all deletions take place at the other end (the front/head)

- First In First Out (FIFO)
- Application of queue in computer science : Job scheduling

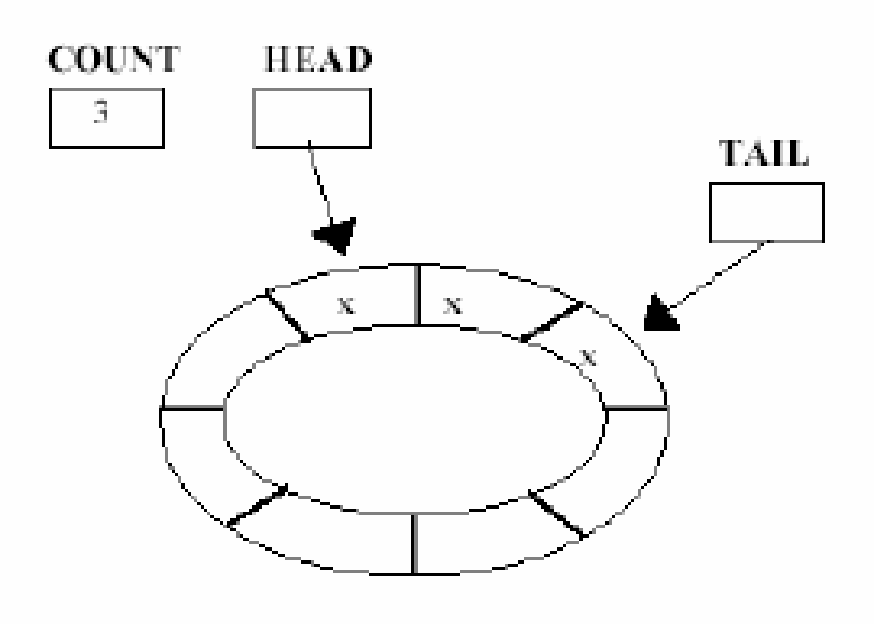
Queue (2)

- Notion of head, tail, element, empty



Queue Variation

- Circular buffered queue



Tree

- A tree is a finite set of one or more nodes such that :

  - There is a specially designated node called the root

  - The remaining nodes are partitioned into n ≥ 0 disjoint sets T1, T2, ... Tn where each set is a tree

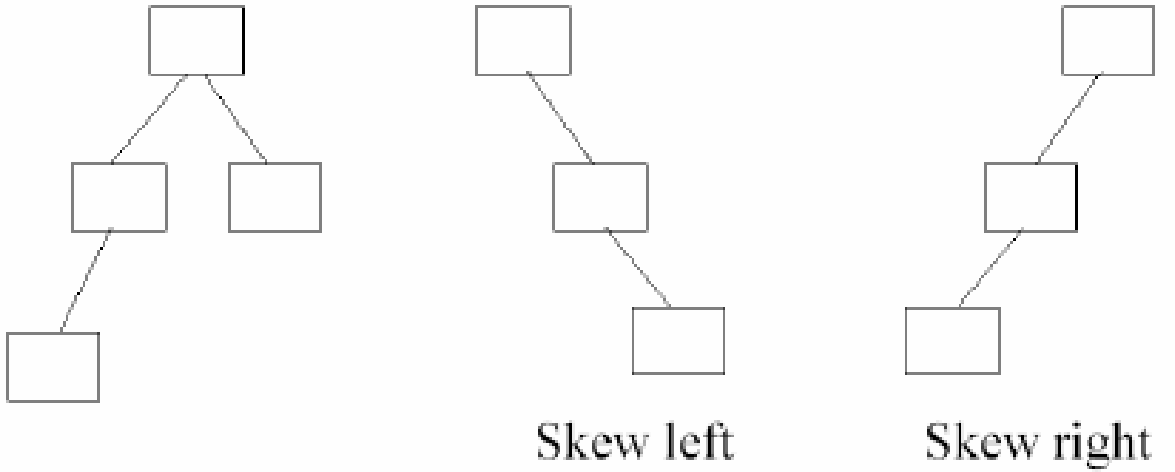
- T1, T2, ... Tn are called the subtree of the root
- N-airy tree, binary tree

Binary Tree (1)

- A tree that any node can have at most two branches
- Distinguish between Left subtree and Right subtree
- A binary tree may have zero nodes (empty tree)
- A binary tree is different object than a tree

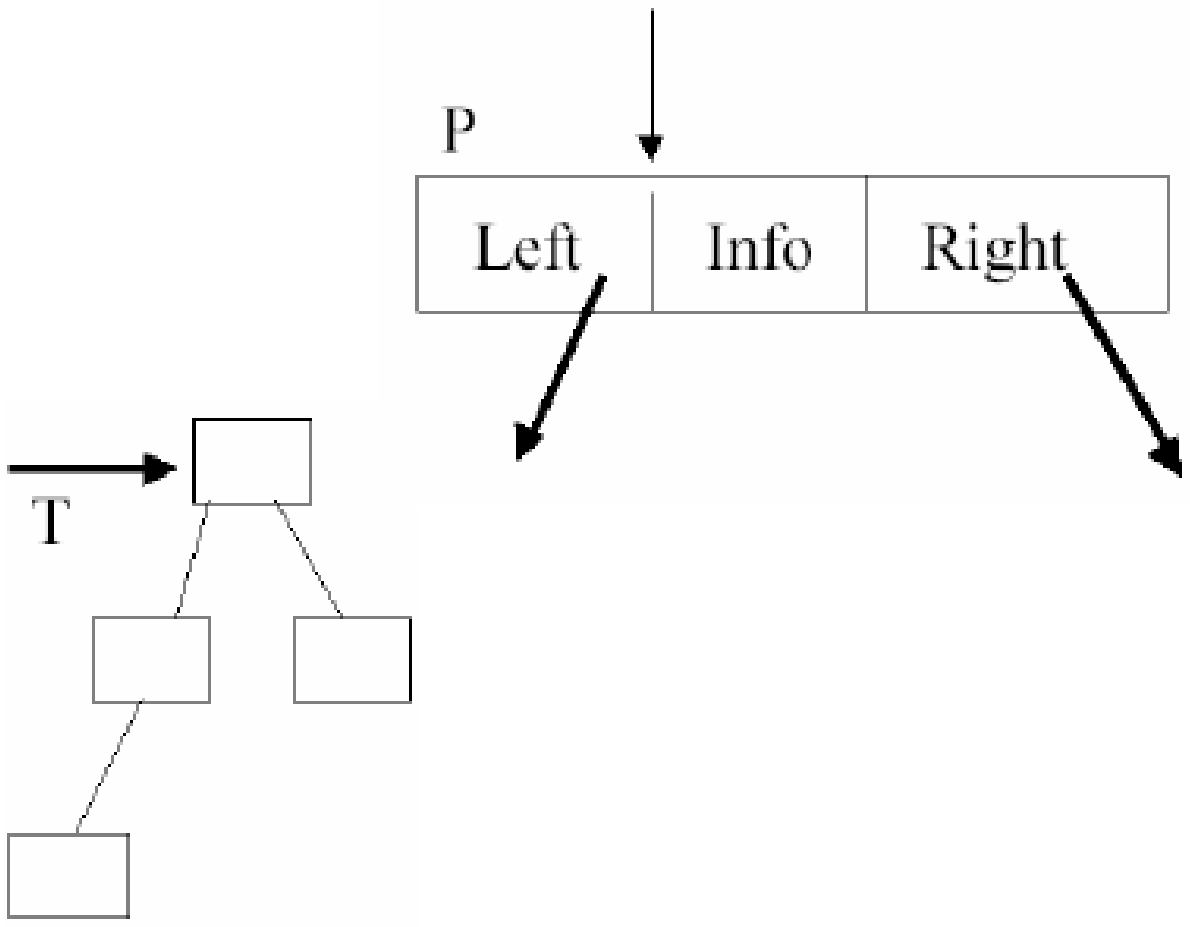
Binary Tree (2)

- A binary tree is a finite set of nodes which is either empty or consists of a root and two disjoint binary tree called the left subtree and the right subtree.



Binary Tree Node

- Left(P)
- Right(P)
- Info(P)
- Root(T)



Binary Tree Representation

- Contiguous representation by array

  - If P is a node, then Left(P)=P*2 and Right(P)=2*P+1

- Linked representation

Binary Tree (3)

- Primitives :
  - Traversal : preorder, inorder, postorder
  - Searching
  - Make Tree

- Variations :
  - Search Tree
  - Balance Tree
  - Threaded Tree

- Method to build a tree :
  - Iterative methods
  - Recursive methods

Data Structure Case Studies

- Polinom
- Occurrence of each letter in a text file
- Memory management
- Multilist
- Representation of N-M relation
- Topological sort