# Unit 10: Structures and Dynamic Memory Allocation

Er. Nipun Thapa

# What is Structure?

- A structure is a key word that create user defined data type in C.
- A structure creates a data type that can be used to group items of possibly different types into a single type.

***How to create a structure?***

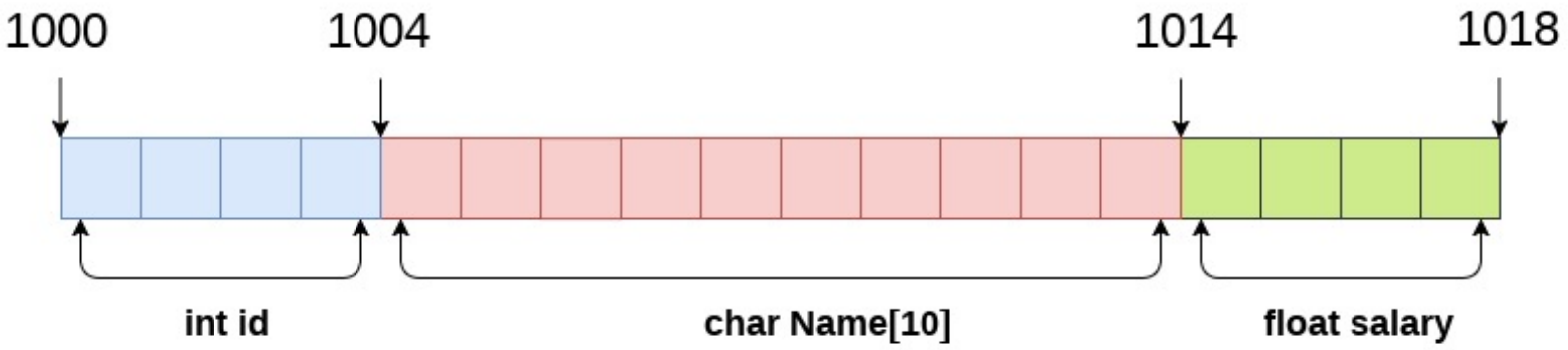
- **struct** keyword is used to create a structure.

```
struct structure_name
{
   data_type member1;
   data_type member2;
   .
   .
   data_type memeberN;
};
```

Example:

```
struct employee
{   int id;
   char name[20];
   float salary;
};
```

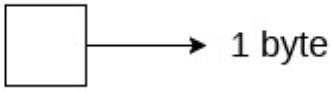
The following image shows the memory allocation of the structure employee that is defined in the above example.



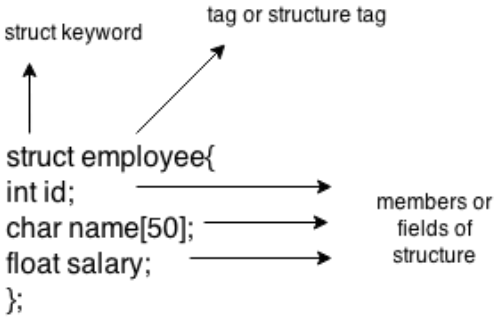

```
struct Employee
{
    int id;
    char Name[10];
    float salary;
} emp;
```

**sizeof (emp) = 4 + 10 + 4 = 18 bytes**

**where;**
**sizeof (int) = 4 byte**
**sizeof (char) = 1 byte**
**sizeof (float) = 4 byte**




Here, **struct** is the keyword; **employee** is the name of the structure; **id**, **name**, and **salary** are the members or fields of the structure. Let's understand it by the diagram given below:



# Declaring structure variable

- We can declare a variable for the structure so that we can access the member of the structure easily.
- There are two ways to declare structure variable:
  1. By struct keyword within main() function
  2. By declaring a variable at the time of defining the structure.

**1st way:**

- Declare the structure variable by struct keyword.
- It should be declared within the main function.

```
struct employee
{   int id;
  char name[50];
  float salary;
};
```

Now write given code inside the main() function.

```
struct employee e1, e2;
```

The variables e1 and e2 can be used to access the values stored in the structure. Here, e1 and e2 can be treated in the same way as the objects in c.

# Declaring structure variable

**2nd way:**

- Let's see another way to declare variable at the time of defining the structure.

```
struct employee
{   int id;
    char name[50];
    float salary;
}e1,e2;
```

*Which approach is good*
*If number of variables are not fixed, use the 1st approach. It provides you the flexibility to declare the structure variable many times.*
*If no. of variables are fixed, use 2nd approach. It saves your code to declare a variable in main() function.*

# Accessing members of the structure

There are two ways to access structure members:

1. By . (member or dot operator)
2. By -> (structure pointer operator)

Let's see the code to access the *id* member of *p1* variable by. (member) operator.

**p1.id**

# Lab8 QN1: C Structure example

```
#include<stdio.h>
#include <string.h>
struct employee
{   int id;
    char name[50];
};
int main( )
{
            struct employee e1;
            printf(“Enter ID:”);
            scanf(“%d”,&e1.id);
            printf(“Enter name:”);
            scanf(“%s”,&e1.name);
            printf( "employee id : %d\n", e1.id);
            printf( "employeename : %s\n", e1.name);
return 0;
}
```

# Lab8 QN2: C Structure example

```
#include<stdio.h>
#include <string.h>
struct employee
{   int id;
    char name[50];
}e1,e2;
int main( )
{
                printf(“Enter ID1:”);
                scanf(“%d”,&e1.id);
                printf(“Enter name1:”);
                scanf(“%s”,&e1.name);
                printf(“Enter ID2:”);
                scanf(“%d”,&e2.id);
                printf(“Enter name2:”);
                scanf(“%s”,&e2.name);
                printf( "employee 1 id : %d\n", e1.id);
                printf( "employee 1 name : %s\n", e1.name);
                printf( "employee 2 id : %d\n", e2.id);
                printf( "employee 2 name : %s\n", e2.name);
return 0;
}
```

# Lab8 QN3: C Structure example

```
#include<stdio.h>
#include <string.h>
struct bill
   {            int id;  char address[200]; float amount;
    }p1,p2;
int main( )
{ p1.id=1;
strcpy(p1.address, ”Mid baneshwor,Kathmandu");
p1.amount=5689.36;
printf("Details of First Person!\n");
printf("Id of first person is: %d\n",p1.id);
printf("Amount due by first person is: %f\n",p1.amount);
printf("Address of first person is: %s\n",p1.address);
p2.id=2;
strcpy(p2.address, ”Kalanki Kathmandu");
p2.amount=5644.36;
printf("Details of Second Person!\n");
printf("Id of Second person is: %d\n",p2.id);
printf("Amount due by second person is: %f\n",p2.amount);
printf("Address of second person is: %s\n",p2.address);
return 0; }
```

# Lab8 QN4: C Structure example

```
#include <stdio.h>
struct employee
{ // structure definition
    char name[20];
    int age;
    int salary;
}
int main()
{
    struct employee person; // declaration of structure variable person
    printf("Enter name,age and salary \n");
    scanf("%s %d %d", person.name, person.age, person.salary);
    printf("%s %d %d \n", person.name, person.age, person.salary);
    return 0;
}
```

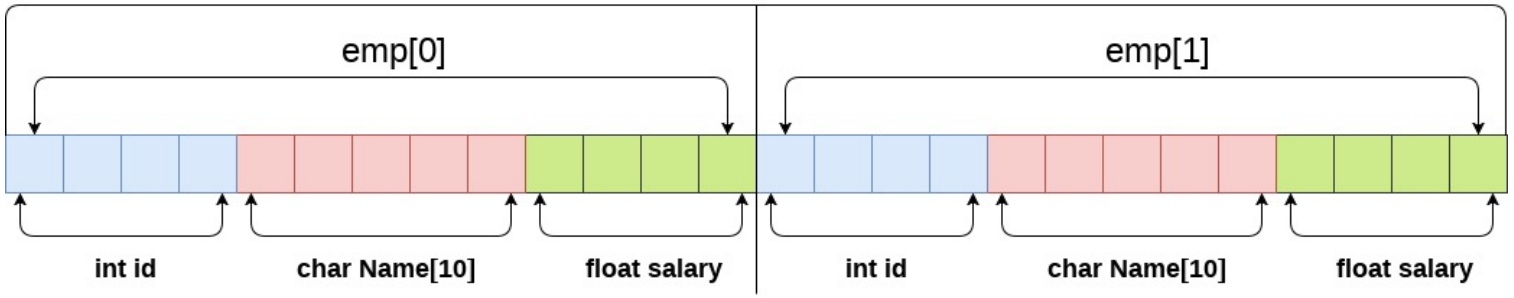
# Lab8 QN5: C Structure example

```
#include<stdio.h>
struct student
{
   char name[20];
   int id;
   float marks;
};
int main()
{
   struct student s1,s2,s3;
   int dummy;
   printf("Enter the name, id, and marks of student 1 ");
   scanf("%s %d %f",s1.name,&s1.id,&s1.marks);
   scanf("%c",&dummy);
   printf("Enter the name, id, and marks of student 2 ");
   scanf("%s %d %f",s2.name,&s2.id,&s2.marks);
   scanf("%c",&dummy);
   printf("Enter the name, id, and marks of student 3 ");
   scanf("%s %d %f",s3.name,&s3.id,&s3.marks);
   scanf("%c",&dummy);
   printf("Printing the details....\n");
   printf("%s %d %f\n",s1.name,s1.id,s1.marks);
   printf("%s %d %f\n",s2.name,s2.id,s2.marks);
   printf("%s %d %f\n",s3.name,s3.id,s3.marks);
  return 0;
}
```

# Array of Structure

- An array of structure in C can be defined as the collection of multiple structures variables where each variable contains information about different entities.
- The array of structures in C are used to store information about multiple entities of different data types. The array of structures is also known as the collection of structures.

**Array of structures**




```
struct employee
{
    int id;
    char name[5];
    float salary;
};
struct employee emp[2];
```

**sizeof (emp) = 4 + 5 + 4 = 13 bytes**

**sizeof (emp[2]) = 26 bytes**

# Lab8 QN5: C Structure example

```
#include<stdio.h>
#include <string.h>
struct student
{
        int rollno;
        char name[10];
};
int main()
{
        int i;
        struct student st[5];
        printf("Enter Records of 5 students");
        for(i=0;i<5;i++)
        {
                printf("\nEnter Rollno:");
                scanf("%d",&st[i].rollno);
                printf("\nEnter Name:");
                scanf("%s",&st[i].name);
        }
        printf("\nStudent Information List:");
        for(i=0;i<5;i++)
        {
                printf("\nRollno:%d, Name:%s",st[i].rollno,st[i].name);
        }
  return 0;
}
```

# Lab8 QN6: Define a structure of employee having data member name,address, age and salary. Take the data of n employee in an array and find the average salary.

```
#include<stdio.h>
 struct employee
{
        char name[20];
        char address[40];
        int age;
        float salary;
};
 int main()
{
        struct employee e[50];
        int n,i;
        float avg=0;
        printf("How many employee:");
        scanf("%d",&n);
        for(i=0;i<n;i++)
        {
        printf("Employee %d info",i+1);
        printf("\nName:");
        scanf("%s",&e[i].name);
        printf("\nAddress:");
        scanf("%s",&e[i].address);
printf("\nAge:");
        scanf("%d",&e[i].age);
        printf("\nSalary:");
        scanf("%d",e[i].salary);
        }

printf("Name\t\tAddress\t\tAge\t\tSalary\n");
  for(i=0;i<n;i++)
    {
        printf("%s\t\t%s\t\t%d\t\t%f\n",e[i].name,e[i].address,e[i].age,e[i].salary");
    }
  for(i=0;i<n;i++)
   {
        avg=avg+e[i].salary;
   }
    avg=avg/i;
    printf("Avg Salary=%f",avg);
   return 0;
}
```

# Structure within another structure(Nested Structure)

```
struct detail
{
        int id;
        float amount;
};
struct info
{
        struct detail each_person;
        int age;
} person_1, person_2;
```

In the above example, we have used two struct types: **detail** and **each_person.** Suppose you want to declare a value of the member **id** for the variable **person_1** then you can do this like below:-

```
person_1.each_person.id = 4;
```

# Nested Structures

- Nesting of structures, is also permitted in C language. Nested structures means, that one structure has another stucture as member variable.

```
struct Student
{
   char name[30];
   int age;
   /* here Address is a structure */
   struct Address
   {
      char[50] locality;
      char[50] city;
      int pincode;
   }addr;
};
```

# Nested Structure

```
struct personal_record
{
        char name[20];
        struct
        {
                int day;
                int month;
                int year;
        }birthday;
        float salary;
}person;
```

| Name | birthday | | | salary |
|---|---|---|---|---|
| | day | month | year | |
| … | … | … | …. | … |
| … | …. | … | .. | … |
| …. | …. | …. | …. | …. |

Here, the structuer persanal_record contain member name birthday which itself is a Structure with 3 member.

**person.name;**
**person.salary;**
**person.birthday.day;**
**person.birthday.month;**
**person.birthday.year;**

# Lab 8 QN7 : Program to demonstrate the use of nested structure 0f 5 strudents

| Name | birthday | | | salary |
|---|---|---|---|---|
| | day | month | year | |

```
#include<stdio.h>
struct personal_record
{
        char name[20];
        struct
        {
                int day;
                int month;
                int year;
        }birthday;
        float salary;
}person[5];
int main()
{
 int i;
 for(i=0;i<5;i++
{  printf("Enter info of %d person record\n",i+1);
   printf("Enter Name:");
   scanf("%s",&person[i].name);
   printf("Enter birthday:");
   scanf("%d",&person[i].dirthday.day);
   printf("Enter birthmonth:");
   scanf("%d",&person[i].birthday.month);
   printf("Enter year:");
   scanf("%d",&person[i].birthday.year);
   printf("Enter salary:");
   scanf("%f",&person[i].salary);
}
 printf("Name\t\tday\t\tmonth\t\tyear\t\tsalary\n");
 for(i=0;i<5;i++)
{
 printf("%s\t\t%d\t\t%d\t\t%d\t\t%f\n",person[i].name,person[i].birthday.day,
person[i].birthday.month,person[i].birthday.year, person[i].salary);
}
return 0;
}
```

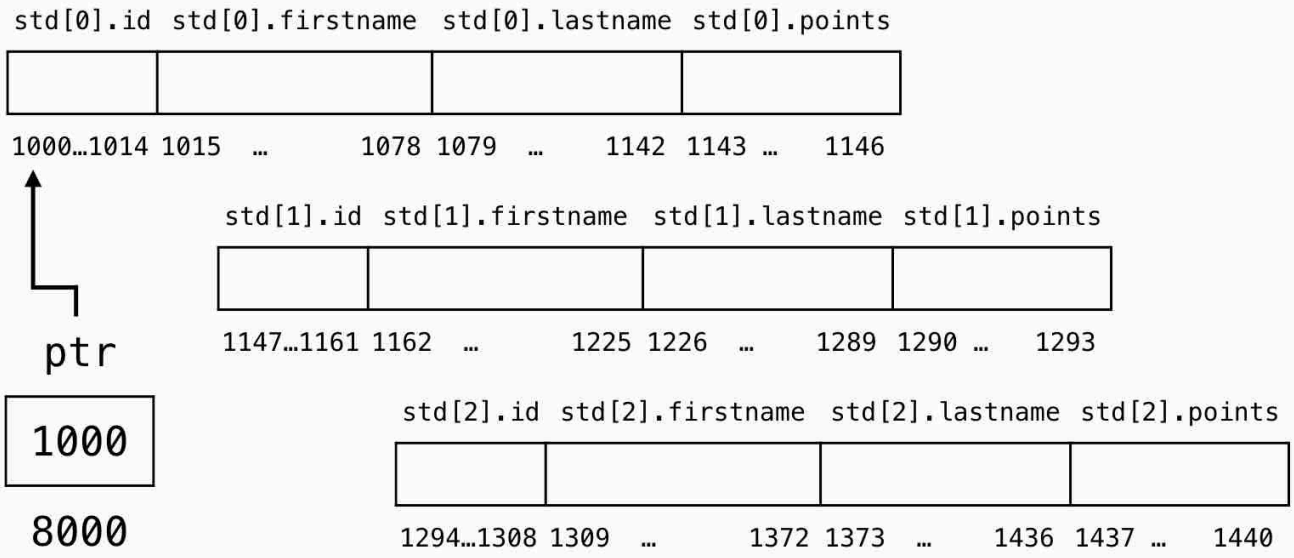
# Pointer to array of structure

```
struct student {
  char id[15];
  char firstname[64];
  char lastname[64];
  float points;
};
```

```
struct student std[3];

struct student *ptr;

ptr = std;
```



## Lab 8 QN 8 . Example of structure pointer

```
#include<stdio.h>
struct point
{           int a;
            float b;
};
int main()
{           struct point p1={3,6.3};
            struct point *p2=&p1;
            printf(”BIM Second semester \n\n");
            printf("First value is: %d\n",p2->a);
            printf("Second value is: %0.1f",p2->b);
            return 0;
}
```

# Lab 8 QN9 : Structure and Pointer

```
#include <stdio.h>
struct person
{
  int age;
  float weight;
};
int main()
{
   struct  person  person1, *personPtr;
   personPtr = &person1;
   printf("Enter age: ");
   scanf("%d", &personPtr->age);
   printf("Enter weight: ");
   scanf("%f", &personPtr->weight);
   printf("Displaying:\n");
   printf("Age: %d\n", personPtr->age);
   printf("weight: %f", personPtr->weight);
   return 0;
}
```

# Pointer to array of structure

```
#include <stdio.h>
int main()
{
 struct student
 {
  char id[15];
  char firstname[64];
  char lastname[64];
  float points;
 };
 struct student std[3],*ptr;
 int i;
 ptr = std;
 for (i = 0; i < 3; i++)
  {
  printf("Enter detail of student #%d\n", (i + 1));
  printf("Enter ID: ");
  scanf("%s", ptr->id);
  printf("Enter first name: ");
  scanf("%s", ptr->firstname);
  printf("Enter last name: ");
  scanf("%s", ptr->lastname);
  printf("Enter Points: ");
  scanf("%f", &ptr->points);
  ptr++;
 }
```

```
 ptr = std;
 for (i = 0; i < 3; i++) {
  printf("\nDetail of student #%d\n", (i + 1));
  printf("\nResult via std\n");
  printf("ID: %s\n", std[i].id);
  printf("First Name: %s\n", std[i].firstname);
  printf("Last Name: %s\n", std[i].lastname);
  printf("Points: %f\n", std[i].points);
  printf("\nResult via ptr\n");
  printf("ID: %s\n", ptr->id);
  printf("First Name: %s\n", ptr->firstname);
  printf("Last Name: %s\n", ptr->lastname);
  printf("Points: %f\n", ptr->points);
  ptr++;
 }
 return 0;
}
```

# Union

- With a union, all members share **the same memory**.
- We can access only one member of union at a time. We can't access all member values at the same time in union.
- But, structure can access all member values at the same time. This is because, Union allocates one common storage space for all its members. Where as Structure allocates storage space for all its members separately.

# Union

```
#include<stdio.h>
union student
{
   char name[20];
   int roll;
   float mark;
};
int main()
{

   union student s;
   printf("Enter name:");
   gets(s.name);
   printf("Enter Roll:");
   scanf("%d",&s.roll);
   s.mark=98.01;
   printf("\nYou record is\n");
   printf("Name:%s", s.name);
   printf("\nRoll:%d ",s.roll);
   printf("\nMarks:%f",s.mark);
   return 0;
}
```

# Union

```
#include <stdio.h>
union Job {
   float salary;
   int workerNo;
} j;

int main() {
   j.salary = 12.3;
      // when j.workerNo is assigned a value,
      // j.salary will no longer hold 12.3
   j.workerNo = 100;
   printf("Salary = %.1f\n", j.salary);
   printf("Number of workers = %d", j.workerNo);
   return 0;
}
```

# Union

```
#include <stdio.h>
#include <string.h>
 union student
{
        char name[20];
        char subject[20];
        float percentage;
}record;
 int main()
{
         strcpy(record.name, "Raju");
        strcpy(record.subject, "Maths");
        record.percentage = 86.50;
         printf(" Name      : %s \n", record.name);
        printf(" Subject    : %s \n", record.subject);
        printf(" Percentage : %f \n", record.percentage);
        return 0;
}
```

o/p

```
Name :
Subject :
Percentage : 86.500000
```

# DIFFERENCE BETWEEN STRUCTURE AND UNION IN C:

| C Structure | C Union |
|---|---|
| Structure allocates storage space for all its members separately. | Union allocates one common storage space for all its members.<br>Union finds that which of its member needs high storage space over other members and allocates that much space |
| Structure occupies higher memory space. | Union occupies lower memory space over structure. |
| We can access all members of structure at a time. | We can access only one member of union at a time. |
| Structure example:<br>struct student<br>{<br>int mark;<br>char name[6];<br>double average;<br>}; | Union example:<br>union student<br>{<br>int mark;<br>char name[6];<br>double average;<br>}; |
| For above structure, memory allocation will be like below.<br>int mark – 2B<br>char name[6] – 6B<br>double average – 8B<br>Total memory allocation = 2+6+8 = 16 Bytes | For above union, only 8 bytes of memory will be allocated since double data type will occupy maximum space of memory over other data types.<br>Total memory allocation = 8 Bytes |

# Difference between unions and structures

```
#include <stdio.h>
union unionJob
{
  //defining a union
  char name[32];
  float salary;
  int workerNo;
} uJob;

struct structJob
{
  char name[32];
  float salary;
  int workerNo;
} sJob;

int main()
{
  printf("size of union = %d bytes", sizeof(uJob));
  printf("\nsize of structure = %d bytes", sizeof(sJob));
  return 0;
}
```

**Output**

```
size of union = 32
size of structure = 40
```

# Why this difference in the size of union and structure variables?

- Here, the size of sJob is 40 bytes because
  - the size of name[32] is 32 bytes
  - the size of salary is 4 bytes
  - the size of workerNo is 4 bytes
- However, the size of uJob is 32 bytes. It's because the size of a union variable will always be the size of its largest element. In the above example, the size of its largest element, (name[32]), is 32 bytes.
- So , With a union, all members share **the same memory**.

# Dynamic Memory Allocation

The concept of **dynamic memory allocation in c language** *enables the C programmer to allocate memory at runtime*.

Dynamic memory allocation in c language is possible by 4 functions of stdlib.h header file.

- malloc()
- calloc()
- realloc()
- free()

| static memory allocation | dynamic memory allocation |
| --- | --- |
| memory is allocated at compile time. | memory is allocated at run time. |
| memory can't be increased while executing program. | memory can be increased while executing program. |
| used in array. | used in linked list. |

# Dynamic Memory Allocation

| | |
|---|---|
| **malloc()** | allocates single block of requested memory. |
| **calloc()** | allocates multiple block of requested memory. |
| **realloc()** | reallocates the memory occupied by malloc() or calloc() functions. |
| **free()** | frees the dynamically allocated memory. |

# malloc() function in C

- The malloc() function allocates single block of requested memory.
- It doesn't initialize memory at execution time, so it has garbage value initially.
- It returns NULL if memory is not sufficient.

**Syntax :**

**ptr=(cast-type*)malloc(byte-size)**

Example :

**ptr = (float*) malloc(100 * sizeof(float));**

- The above statement allocates 400 bytes of memory.
- It's because the size of float is 4 bytes. And, the pointer ptr holds the address of the first byte in the allocated memory.
- The expression results in a NULL pointer if the memory cannot be allocated.

# calloc()

- The calloc() function allocates multiple block of requested memory.
- It initially initialize all bytes to zero.
- It returns NULL if memory is not sufficient.

**Syntax:**

**ptr=(cast-type*)calloc(number, byte-size)**

**Example:**

**ptr = (float*) calloc(25, sizeof(float));**

The above statement allocates contiguous space in memory for 25 elements of type float.

# realloc() function in C

- If memory is not sufficient for malloc() or calloc(), we can reallocate the memory by realloc() function.
- In short, it changes the memory size.
- Let's see the syntax of realloc() function.

**ptr=realloc(ptr, new-size)**

# free() function in C

- The memory occupied by malloc() or calloc() functions must be released by calling free() function.
- Otherwise, it will consume memory until program exit.

Let's see the syntax of free() function.

**free(ptr)**

# How to deallocate memory without using free() in C?

- Standard library function realloc() can be used to deallocate previously allocated memory. Below is function declaration of “realloc()” from “stdlib.h”

# Linked list (-> operator, creating, displaying, searching)

- There are various linked list operations that allow us to perform different actions on linked lists. For example, the insertion operation adds a new element to the linked list.
- Here's a list of basic linked list operations that we will cover in this article.
  - **Traversal** - access each element of the linked list
  - **Insertion** - adds a new element to the linked list
  - **Deletion** - removes the existing elements
  - **Search** - find a node in the linked list
  - **Sort** - sort the nodes of the linked list

# Finished
# Unit 10