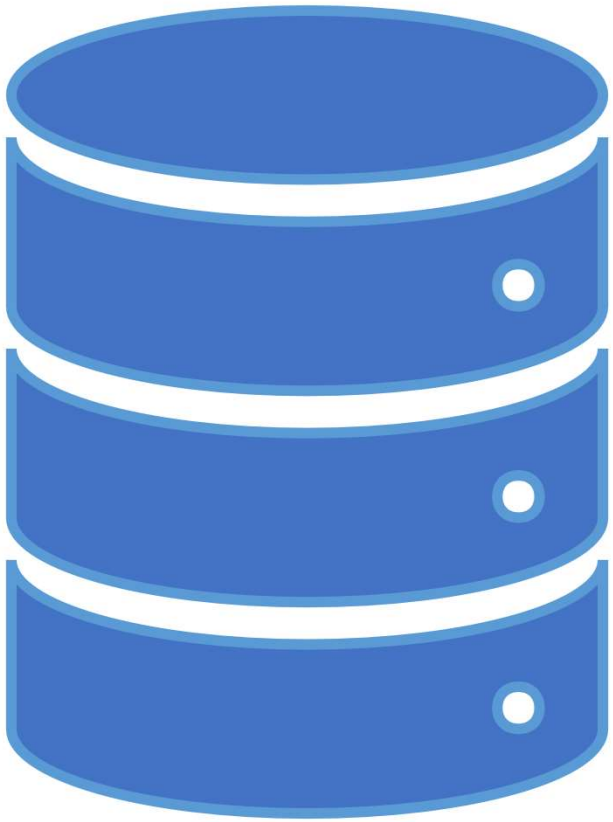
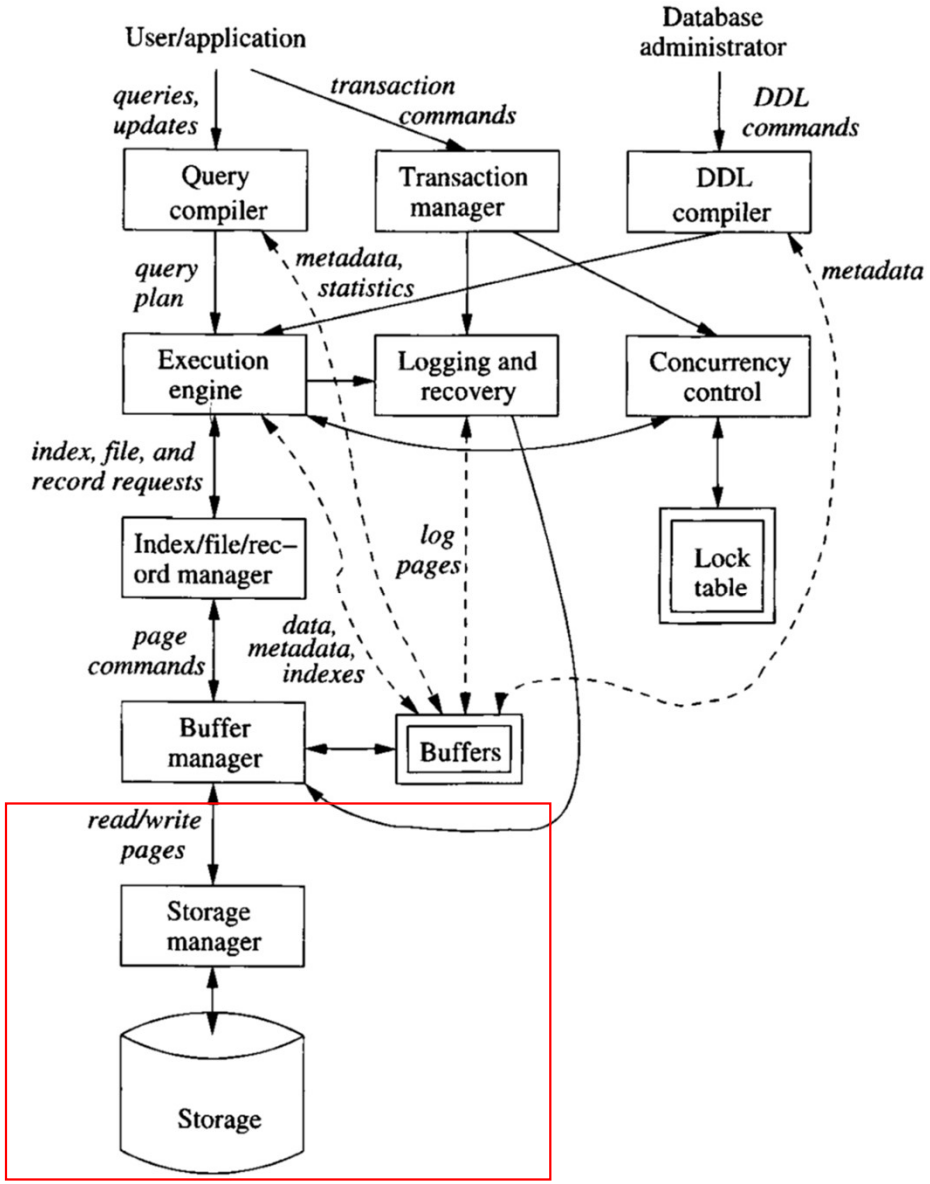


Database Programming

Lecture 6: Working with pointers

Dr. Samir Tartir

Where to start?



The meaning of &

```
#include <stdio.h>
int main () {

   int  var1;      // Create a variable
   char var2[10];  // Create an array of type char

   printf("Address of var1 variable: %x\n", &var1);
   printf("Address of var2 variable: %x\n", &var2);

   return 0;
}
```

%x: print hexadecimal number

The output is:

Address of var1 variable: bff5a400

Address of var2 variable: bff5a3f6

Pointer arithmetic

- In arrays of C programming, name of the array always ***points*** to the first element of an array
- The address of first element of an array is &arr[0]
- &arr[0] is equivalent to arr
  - Similarly &arr[1] is equivalent to (arr+1)
  - Similarly &arr[2] is equivalent to (arr+2)……

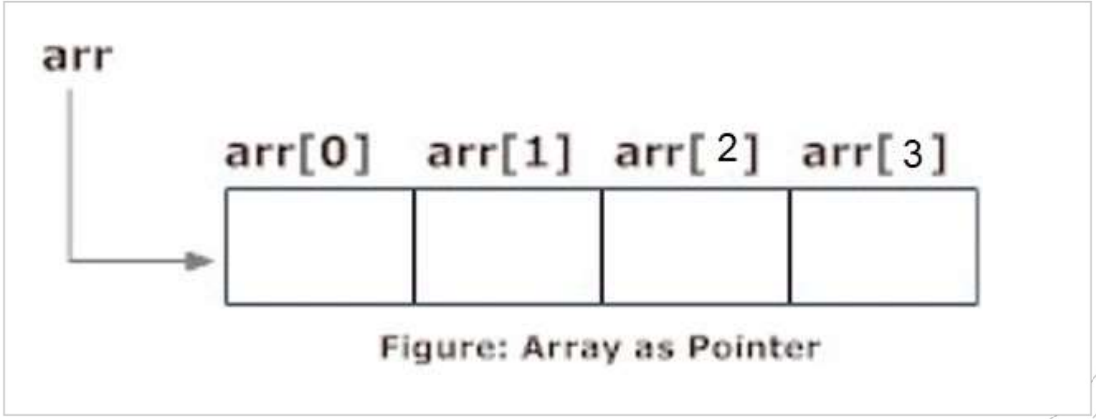


Figure: Array as Pointer

int arr[4];

Example

- Note: %x: lower-case hexadecimal

```
#include <stdio.h>
int main(){
   int arr[4] = {100, 200, 300, 400}; // Declare an array of type int
   printf("Address of arr[0]: %x\n", arr);
   printf("Address of arr[1]: %x\n", (arr+1));
   printf("Address of arr[2]: %x\n", (arr+2));
   printf("Address of arr[3]: %x\n", (arr+3));
return 0;
}
```

The output is:

Address of arr[0]: 4ebd4870
Address of arr[1]: 4ebd4874
Address of arr[2]: 4ebd4878
Address of arr[3]: 4ebd487c

Or, we can use a for loop

```
#include <stdio.h>
int main(){
   int arr[4] = {100, 200, 300, 400};
   for(int i = 0; i <4; i++){
       printf("Address of arr[%d]: %x\n", i, (arr+i));
   }
return 0;
}
```

The output is:

Address of arr[0]: 4ebd4870
Address of arr[1]: 4ebd4874
Address of arr[2]: 4ebd4878
Address of arr[3]: 4ebd487c

Or, we can use a for loop

```
#include <stdio.h>
int main(){
 int arr[4] = {100, 200, 300, 400};
 printf("Address of arr[0]: %x\n", arr);
 printf("Address of arr[1]: %x\n", (arr+1));
 printf("Address of arr[2]: %x\n", (arr+2));
 printf("Address of arr[3]: %x\n", (arr+3));

 //3 or we can say
 for(int i = 0; i <4; i++){
    printf("Address of arr[%d]: %x\n", i, (arr+i));
 }

//another way
 for(int i = 0; i <4; i++){
    printf("Address of arr[%d]: %x\n", i, &arr[i]);
  }
return 0;
}
```

The output is:

```
Address of arr[0]: 4ebd4870
Address of arr[1]: 4ebd4874
Address of arr[2]: 4ebd4878
Address of arr[3]: 4ebd487c

Address of arr[0]: 4ebd4870
Address of arr[1]: 4ebd4874
Address of arr[2]: 4ebd4878
Address of arr[3]: 4ebd487c

Address of arr[0]: 4ebd4870
Address of arr[1]: 4ebd4874
Address of arr[2]: 4ebd4878
Address of arr[3]: 4ebd487c
```

Example

- What is the output here knowing that the address of the first element is 28ff44.
- Note: %x: lower-case hexadecimal

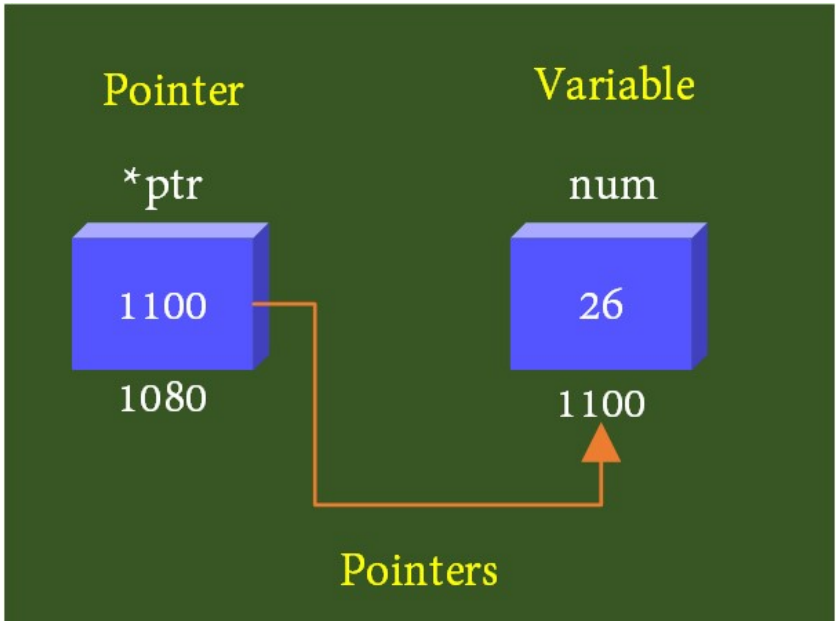
```
#include <stdio.h>
int main(){
   char c[4];
   int i;
   for(i=0;i<4;++i){
      printf("Address of c[%d]=%x\n",i,&c[i]);
   }
   return 0;
}
```

The output is:

Address of c[0]=28ff44
Address of c[1]=28ff45
Address of c[2]=28ff46
Address of c[3]=28ff47

Pointers

- One of the sophisticated C programming fields.
- Based on the *indirection* principle.
- *ptr* holds the address of the variable *num*



How to define pointers

All you need to do is:

1. Define the original variable
2. Define the pointer to indirectly access the variable
3. Connect them

Example:

```
int count =10;
int  *int_pointer; // note the *
int_pointer = &count; //note the &
```

Example

```
#include <stdio.h>
int main(){
    int num =10;
    int  *p1;
    p1 = &num;
    printf("num = %d, &num= %x, p1=%x, *p1=%d \n",num, &num, p1, *p1);
    return 0;
}
```

The output is:

num = 10, &num= f6ad1b68, p1=f6ad1b68, *p1=10

First Program

```
#include <stdio.h>
int main (void) {
   int   count = 10, x;
   int   *int_pointer;
   int_pointer = &count;
   x = *int_pointer;
   printf ("count = %i, x = %i\n", count, x);
   return 0;
}
```

The output is:

count = 10, x = 10

Example

```
#include <stdio.h>
int main (void)
{
   char  c = 'Q';
   char  *char_pointer = &c;
   printf ("%c %c\n", c, *char_pointer);
   c = '/';
   printf ("%c %c\n", c, *char_pointer);
   *char_pointer = '(';
   printf ("%c %c\n", c, *char_pointer);
   return 0;
}
```

The output is:

Q Q

/ /

((

What is the output of this program

```
#include <stdio.h>
int main (void)
{
        int  i1, i2;
        int  *p1, *p2;
        i1 = 5;
        p1 = &i1;
        i2 = *p1 / 2 + 10;
        p2 = p1;
        printf ("i1 = %i, i2 = %i, *p1 = %i, *p2 = %i\n", i1, i2, *p1, *p2);
        return 0;
}
```

The output is:

i1 = 5, i2 = 12, *p1 = 5, *p2 = 5

Pointer’s Subtraction

```
#include<stdio.h>
int main()
{
   int num , *ptr1 ,*ptr2 ;
   ptr1 = &num ;
   ptr2 = ptr1 + 2 ;
   printf("%d",ptr2 - ptr1);
   return(0);
}
```

The output is:

2

Example

```
#include <stdio.h>
int main () {
   int  var = 20;   // actual variable declaration
   int  *ip;        // pointer variable declaration

   ip = &var;  // store address of var in pointer variable
   printf("Address of var variable: %x\n", &var  );

   /* address stored in pointer variable */
   printf("Address stored in ip variable: %x\n", ip );

  /* access the value using the pointer */
   printf("Value of *ip variable: %d\n", *ip );
   return 0;
}
```

Var address is bffd8b3c

The output is:

Address of var variable: bffd8b3c
Address stored in ip variable: bffd8b3c
Value of *ip variable: 20

Another example

```
#include <stdio.h>
int main(){
   int data[5], i;
   printf("Enter elements: ");
   for(i=0;i<5;++i)
     scanf("%d",data+i);
   printf("You entered: ");
   for(i=0;i<5;++i)
      printf("%d\n",*(data+i));
   return 0;
}
```

The output is:

```
Enter elements: 1
2
3
5
4
You entered: 1
2
3
5
4
```

Of course, we can use the array name

```
#include <stdio.h>
int main () {
   /* an array with 5 elements */
   float balance[5] = {1000.0, 2.0, 3.4, 17.0, 50.0};
   float *p;
   int i;

   p = balance;

   /* output each array element's value */
   printf( "Array values using pointer\n");
   for ( i = 0; i < 5; i++ ) {
      printf("*(p + %d) : %f\n",  i, *(p + i) );
   }

   printf( "Array values using balance as address\n");
   for ( i = 0; i < 5; i++ ) {
      printf("*(balance + %d) : %f\n",  i, *(balance + i) );
   }
   return 0;
}
```

The output is:

Array values using pointer
*(p + 0) : 1000.000000
*(p + 1) : 2.000000
*(p + 2) : 3.400000
*(p + 3) : 17.000000
*(p + 4) : 50.000000
Array values using balance as address
*(balance + 0) : 1000.000000
*(balance + 1) : 2.000000
*(balance + 2) : 3.400000
*(balance + 3) : 17.000000
*(balance + 4) : 50.000000

Passing Arguments to Functions

Call by value

Call by reference

Call by value

```
#include<stdio.h>
void swap(int n1, int n2)
{
    int temp;
    temp = n1;
    n1 = n2;
    n2 = temp;
    printf("\nNumber 1 : %d",n1);
    printf("\nNumber 2 : %d",n2);
}

int main() {
    int num1=50, num2=70;
    swap(num1,num2);

    printf("\nNumber 1 : %d",num1);
    printf("\nNumber 2 : %d",num2);

    return(0);
}
```

The output is:

Number 1 : 70

Number 2 : 50

Number 1 : 50

Number 2 : 70

Call by reference

```
#include<stdio.h>
void swap(int *n1, int *n2)
{
    int temp;
    temp  = *n1;
    *n1 = *n2;
    *n2 = temp;
    printf("\nNumber 1 : %d",*n1);
    printf("\nNumber 2 : %d",*n2);}

int main() {
    int num1=50, num2=70;
    swap(&num1,&num2);

    printf("\nNumber 1 : %d",num1);
    printf("\nNumber 2 : %d",num2);

    return(0);
}
```

The output is:

Number 1 : 70

Number 2 : 50

Number 1 : 70

Number 2 : 50

Calling summary

Point	Call by Value	Call by Reference
Copy	Duplicate Copy of Original Parameter is Passed	Actual Copy of Original Parameter is Passed
Modification	No effect on Original Parameter after modifying parameter in function	Original Parameter gets affected if value of parameter changed inside function

Passing pointer to a function

```
#include <stdio.h>
double getAverage(int arr[], int size){
   int  i, sum = 0;
   double avg;
   for (i = 0; i < size; ++i)
      sum += arr[i];
   avg = (double)sum / size;
   return avg;
}

int main (){
   int balance[5] = {1000, 2, 3, 17, 50};
   double avg;
   avg = getAverage( balance, 5 ) ;
   printf("Average value is: %f\n", avg );
   return 0;
}
```

The output is:

Average value is: 214.40000


Pointers and
Structures

Pointers to Struct

- The dot in pointers (*x).y is replaced by ->
- Must know the difference between structures containing pointers and pointer structures

```
int main(){
    struct date {
        int  month;
        int  day;
        int  year;
    };
    struct date  today;
    struct date *datePtr;
    today.month = 11;
    today.day = 8;
    today.year = 2022;
    printf ("Today's date is %i/%i/%.2i.\n", today.month, today.day, today.year % 100);
    datePtr = &today;
    datePtr->month = 12;
    datePtr->day = 12;
    datePtr->year = 2024;
    printf ("datePtr's date is %i/%i/%.2i.\n", datePtr->month, datePtr->day, datePtr->year % 100);
```

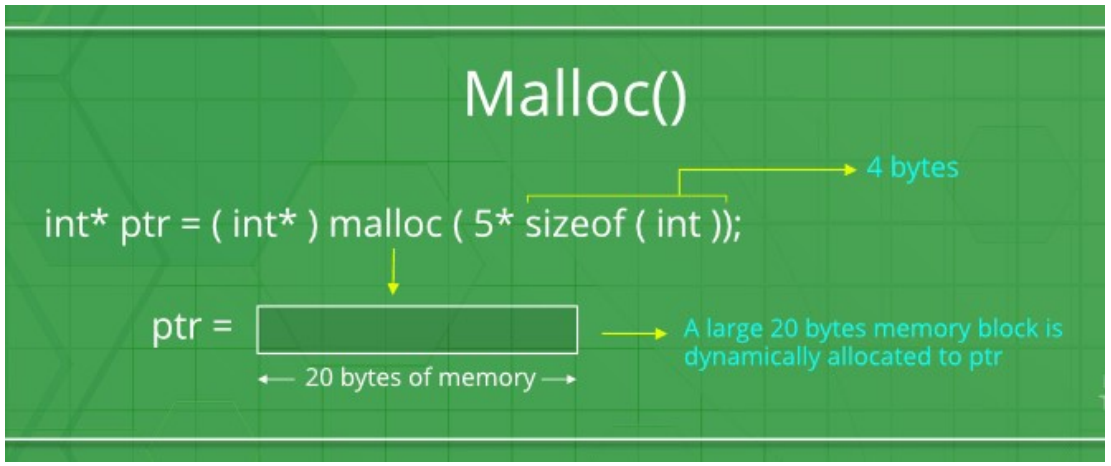
The output is:

Today's date is 11/8/22.

datePtr's date is 12/12/24.

Dynamic memory allocation

- The **“malloc”** or **“memory allocation”** method in C is used to dynamically allocate a single large block of memory with the specified size.
- Syntax***: type *name = (cast-type*) malloc(byte-size);***
- Example: ***int *ptr = (int*) malloc(sizeof(int));***
- To free allocated memory, use the command free
  - ***free(ptr);***



Pointers to struct with malloc

```
#include <stdio.h>
#include <stdlib.h>
struct point {
  int x;
  int y;
};

int main() {
  struct point *pt;

  // Memory allocation for noOfRecords structures
  pt = (struct point *)malloc(sizeof(struct point));
  pt -> x = 3;
  pt -> y = 5;

  printf("X coordinate of the point is: %d\n",pt->x);
  printf("Y coordinate of the point is: %d",pt->y);


  free(pt);
  return 0;
}
```

Output

X coordinate of the point is: 3

Y coordinate of the point is: 5

Structures containing pointers

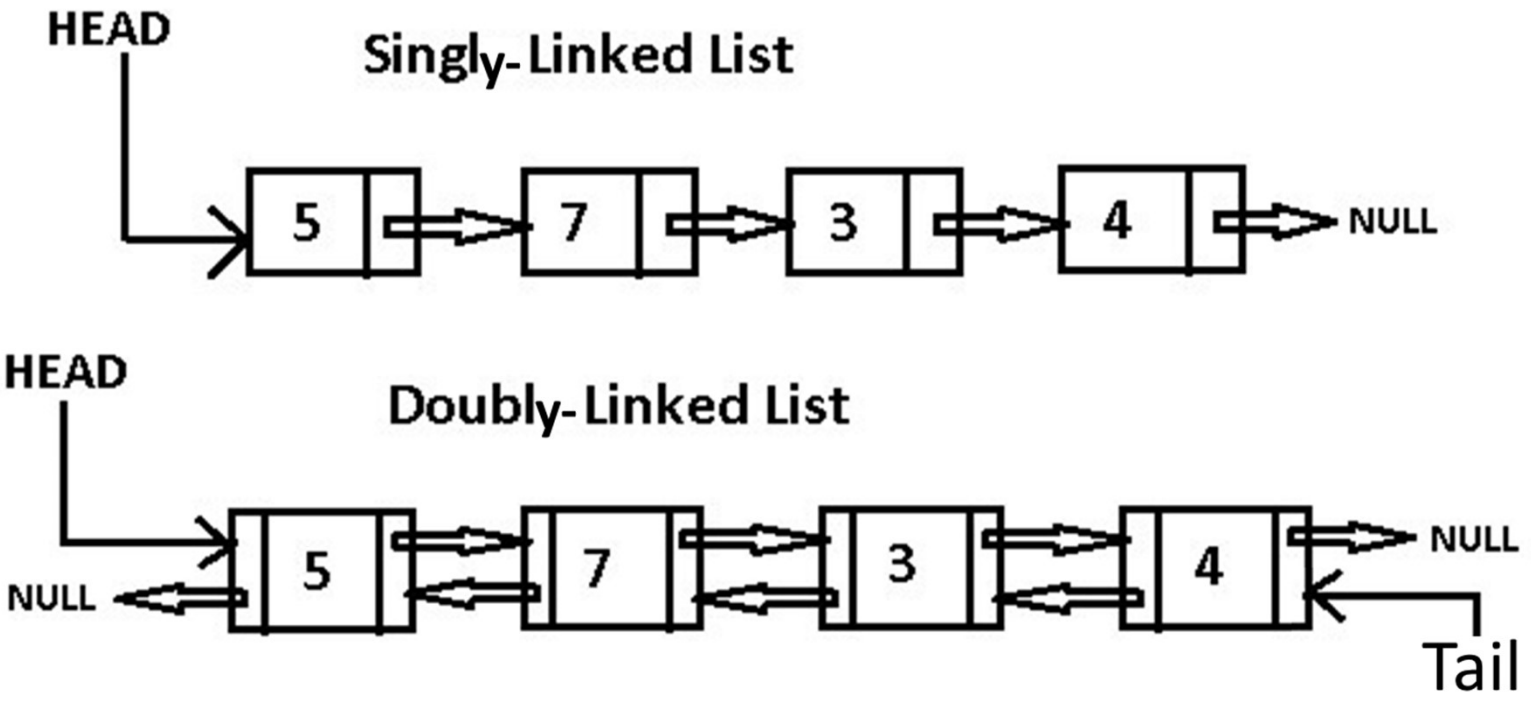
```
#include <stdio.h>
int main (void)
{
struct  intPtrs
{
   int  *p1;
   int  *p2;
};
struct intPtrs  pointers;
int  i1 = 100, i2;
pointers.p1 = &i1;
pointers.p2 = &i2;
*pointers.p2 = -97;
printf ("i1 = %i, *pointers.p1 = %i\n", i1, *pointers.p1);
printf ("i2 = %i, *pointers.p2 = %i\n", i2, *pointers.p2);
return 0;
}
```

The output is:

i1 = 100, *pointers.p1 = 100

i2 = -97, *pointers.p2 = -97

Linked list vs double linked list

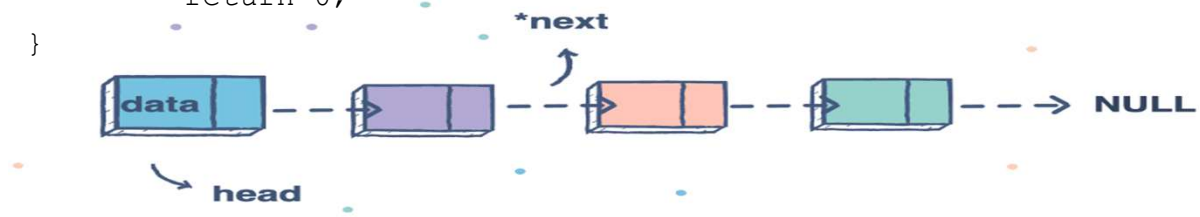


Singly-Linked lists

```
#include <stdio.h>
int main (void)
{
        struct  node
        {
           int  value;
            struct node  *next;
        };
        struct node *head;
        struct node n1, n2, n3;
        int  i;
        n1.value = 100;
        n2.value = 200;
        n3.value = 300;
        head = &n1;
        n1.next = &n2;
        n2.next = &n3;
        i = n1.next-> value;
        printf ("%i  ", i);
        printf ("%i\n", n2.next->value);
        return 0;
}
```

The output is:

200 300



Acknowledgment

Slides were prepared by Eng. Lina Hammad
Modified by Dr. Ala’a Al-Habashna