* Pointers
* Find Largest Element Using Dynamic Memory Allocation - calloc()
* Write a C program to find sum of n elements entered by user. To perform this program, allocate memory dynamically using malloc() function.
* Demonstrate the Dynamic Memory Allocation for Structure

### Example To Demonstrate Working of Pointers

#include <stdio.h>

int main(){

int\* pc;

  int c;

c=22;

printf("Address of c:%u\n",&c);

printf("Value of c:%d\n\n",c);

pc=&c;

printf("Address of pointer pc:%u\n",pc);

printf("Content of pointer pc:%d\n\n",\*pc);

c=11;

printf("Address of pointer pc:%u\n",pc);

printf("Content of pointer pc:%d\n\n",\*pc);

\*pc=2;

printf("Address of c:%u\n",&c);

printf("Value of c:%d\n\n",c);

return 0;

}

Address of c: 2686784

Value of c: 22

Address of pointer pc: 2686784

Content of pointer pc: 22

Address of pointer pc: 2686784

Content of pointer pc: 11

Address of c: 2686784

Value of c: 2

Pointer arithmetic

#include <stdio.h>

const int MAX = 3;

int main () {

int var[] = {10, 100, 200};

int i, \*ptr;

/\* let us have array address in pointer \*/

ptr = &var;

for ( i = 0; i < MAX; i++) {

printf("Address of var[%d] = %x\n", i, ptr );

printf("Value of var[%d] = %d\n", i, \*ptr );

/\* move to the next location \*/

ptr++;

}

return 0;

}

Address of var[0] = bf882b30

Value of var[0] = 10

Address of var[1] = bf882b34

Value of var[1] = 100

Address of var[2] = bf882b38

Value of var[2] = 200

**rand() and srand() in C**

**rand ()**

rand() function is used in C to generate random numbers. If we generate a sequence of random number with rand() function, it will create the same sequence again and again every time program runs. Say if we are generating 5 random numbers in C with the help of rand() in a loop, then every time we compile and run the program our output must be the same sequence of numbers.  
**Syntax:**

**int rand(void):**

returns a pseudo-random number in the range of 0 to RAND\_MAX.

**RAND\_MAX:** is a constant whose default value may vary

between implementations but it is granted to be at least 32767.

|  |
| --- |
| // C program to generate random numbers  #include <stdio.h>  #include <stdlib.h>    // Driver program  int main(void)  {      // This program will create same sequence of      // random numbers on every program run        for(int i = 0; i<5; i++)          printf(" %d ", rand());      return 0;  } |

**NOTE:** This program will create same sequence of random numbers on every program run.  
Output 1:

453 1276 3425 89

Output 2:

453 1276 3425 89

Output n:

453 1276 3425 89

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***s**

**srand()**

The srand() function sets the starting point for producing a series of pseudo-random integers. If srand() is not called, the rand() seed is set as if srand(1) were called at program start. Any other value for seed sets the generator to a different starting point.  
**Syntax:**

**void srand( unsigned seed ):**

Seeds the pseudo-random number generator used by rand()

with the value seed.

**Note:** The pseudo-random number generator should only be seeded once, before any calls to rand(), and the start of the program. It should not be repeatedly seeded, or reseeded every time you wish to generate a new batch of pseudo-random numbers.  
Standard practice is to use the result of a call to **srand(time(0))** as the seed. However, time() returns a time\_t value which vary everytime and hence the pseudo-random number vary for every program call.

|  |
| --- |
| // C program to generate random numbers  #include <stdio.h>  #include <stdlib.h>  #include<time.h>    // Driver program  int main(void)  {      // This program will create different sequence of      // random numbers on every program run        // Use current time as seed for random generator      srand(time(0));        for(int i = 0; i<5; i++)          printf(" %d ", rand());        return 0;  } |

**NOTE:** This program will create different sequence of random numbers on every program run.  
Output 1:

453 1432 325 89

Output 2:

8976 21234 45 8975

Output n:

563 9873 12321 24132

**How srand() and rand() are related to each other?**

srand() sets the seed which is used by rand to generate “random” numbers. If you don’t call srand before your first call to rand, it’s as if you had called srand(1) to set the seed to one.  
In short, **srand() — Set Seed for rand() Function**

**Generate random numbers in specific range**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

int main() {

int i,n,upper,lower;

srand( (unsigned)time( NULL ) );

printf("Enter the lower and upper limit : \n");

scanf("%d%d",&lower,&upper);

printf("Ten random numbers in [1,10]\n");

for (i = 1; i <= 10; i++)

{

n = rand() % (upper-lower+1)+lower;

printf("%d\n", n);

}

return 0;

}

**Example #1: LCM using while Loop and if Statement**

#include <stdio.h>

int main()

{

int n1, n2, minMultiple;

printf("Enter two positive integers: ");

scanf("%d %d", &n1, &n2);

// maximum number between n1 and n2 is stored in minMultiple

minMultiple = (n1>n2) ? n1 : n2;

// Always true

while(1)

{

if( minMultiple%n1==0 && minMultiple%n2==0 )

{

printf("The LCM of %d and %d is %d.", n1, n2,minMultiple);

break;

}

++minMultiple;

}

return 0;

}

**Example #2: GCD Using while loop and if...else Statement**

#include <stdio.h>

int main()

{

int n1, n2;

printf("Enter two positive integers: ");

scanf("%d %d",&n1,&n2);

while(n1!=n2)

{

if(n1 > n2)

n1 -= n2;

else

n2 -= n1;

}

printf("GCD = %d",n1);

return 0;

}

Find Largest Element Using Dynamic Memory Allocation - calloc()

int main()

{

int i, num;

float \*data;

printf("Enter total number of elements(1 to 100): ");

scanf("%d", &num);

// Allocates the memory for 'num' elements.

data = (float\*) calloc(num, sizeof(float));

if(data == NULL)

{

printf("Error!!! memory not allocated.");

exit(0);

}

printf("\n");

// Stores the number entered by the user.

for(i = 0; i < num; ++i)

{

printf("Enter Number %d: ", i + 1);

scanf("%f", data + i);

}

// Loop to store largest number at address data

for(i = 1; i < num; ++i)

{

// Change < to > if you want to find the smallest number

if(\*data < \*(data + i))

\*data = \*(data + i);

}

### Example #1: Using C malloc() and free()

**Write a C program to find sum of n elements entered by user. To perform this program, allocate memory dynamically using malloc() function.**

#include <stdio.h>

#include <stdlib.h>

int main()

{

int num, i, \*ptr, sum = 0;

printf("Enter number of elements: ");

scanf("%d", &num);

ptr = (int\*) malloc(num \* sizeof(int)); //memory allocated using malloc

if(ptr == NULL)

{

printf("Error! memory not allocated.");

exit(0);

}

printf("Enter elements of array: ");

for(i = 0; i < num; ++i)

{

scanf("%d", ptr + i);

sum += \*(ptr + i);

}

printf("Sum = %d", sum);

free(ptr);

return 0;

}

### Example #3: Using realloc()

#include <stdio.h>

#include <stdlib.h>

int main()

{

int \*ptr, i , n1, n2;

printf("Enter size of array: ");

scanf("%d", &n1);

ptr = (int\*) malloc(n1 \* sizeof(int));

printf("Address of previously allocated memory: ");

for(i = 0; i < n1; ++i)

printf("%u\t",ptr + i);

printf("\nEnter new size of array: ");

scanf("%d", &n2);

ptr = realloc(ptr, n2);

for(i = 0; i < n2; ++i)

printf("%u\t", ptr + i);

return 0;

}

## Example: Demonstrate the Dynamic Memory Allocation for Structure

#include <stdio.h>

#include<stdlib.h>

struct course

{

int marks;

char subject[30];

};

int main()

{

struct course \*ptr;

int i, noOfRecords;

printf("Enter number of records: ");

scanf("%d", &noOfRecords);

// Allocates the memory for noOfRecords structures with pointer ptr pointing to the base address.

ptr = (struct course\*) malloc (noOfRecords \* sizeof(struct course));

for(i = 0; i < noOfRecords; ++i)

{

printf("Enter name of the subject and marks respectively:\n");

scanf("%s %d", &(ptr+i)->subject, &(ptr+i)->marks);

}

printf("Displaying Information:\n");

for(i = 0; i < noOfRecords ; ++i)

printf("%s\t%d\n", (ptr+i)->subject, (ptr+i)->marks);

return 0;

}