C Pointers

The pointer in C language is a variable, it is also known as locator or indicator that points to an address of a value.

Advantage of pointer

1) Pointer reduces the code and improves the performance, it is used to retrieving strings, trees etc. and used with arrays, structures and functions.

2) We can return multiple values from a function using the pointer.

3) It makes you able to access any memory location in the computer's memory.

Usage of pointer

There are many usage of pointers in c language.

1) Dynamic memory allocation

In c language, we can dynamically allocate memory using malloc() and calloc() functions where pointer is used.

2) Arrays, Functions and Structures

Pointers in c language are widely used in arrays, functions and structures. It reduces the code and improves the performance.

Symbols used in pointer

Symbol Name Description

& (ampersand sign) address of operator determines the address of a variable.

\* (asterisk sign) indirection operator accesses the value at the address.

Address Of Operator

The address of operator '&' returns the address of a variable. But, we need to use %u to display the address of a variable.

1. #include <stdio.h>

2. #include <conio.h>

3. void main(){

4. int number=50;

5. clrscr();

6. printf("value of number is %d, address of number is %u",number,&number);

7. getch();

8. }

Output

value of number is 50, address of number is fff4

Declaring a pointer

The pointer in c language can be declared using \* (asterisk symbol).

1. int \*a;//pointer to int

2. char \*c;//pointer to char

Pointer example

An example of using pointers printing the address and value is given below.

As you can see in the above figure, pointer variable stores the address of number variable i.e. fff4. The value of the number variable is 50. But the address of pointer variable p is aaa3.

By the help of \* (indirection operator), we can print the value of pointer variable p.

Let's see the pointer example as explained for above figure.

1. #include <stdio.h>

2. #include <conio.h>

3. void main(){

4. int number=50;

5. int \*p;

6. clrscr();

7. p=&number;//stores the address of number variable

8.

9. printf("Address of number variable is %x \n",&number);

10. printf("Address of p variable is %x \n",p);

11. printf("Value of p variable is %d \n",\*p);

12.

13. getch();

14. }

Output

Address of number variable is fff4

Address of p variable is fff4

Value of p variable is 50

NULL Pointer

A pointer that is not assigned any value but NULL is known as NULL pointer. If you don't have any address to be specified in the pointer at the time of declaration, you can assign NULL value. It will a better approach.

int \*p=NULL;

In most the libraries, the value of pointer is 0 (zero).

Pointer Program to swap 2 numbers without using 3rd variable

1. #include<stdio.h>

2. #include<conio.h>

3. void main(){

4. int a=10,b=20,\*p1=&a,\*p2=&b;

5. clrscr();

6.

7. printf("Before swap: \*p1=%d \*p2=%d",\*p1,\*p2);

8. \*p1=\*p1+\*p2;

9. \*p2=\*p1-\*p2;

10. \*p1=\*p1-\*p2;

11. printf("\nAfter swap: \*p1=%d \*p2=%d",\*p1,\*p2);

12.

13. getch();

14. }

Output

Before swap: \*p1=10 \*p2=20

After swap: \*p1=20 \*p2=10

C Pointer to Pointer

In C pointer to pointer concept, a pointer refers to the address of another pointer.

In c language, a pointer can point to the address of another pointer which points to the address of a value. Let's understand it by the diagram given below:

Let's see the syntax of the pointer to the pointer.

1. int \*\*p2;

C pointer to pointer example

Let's see an example where one pointer points to the address of another pointer.

As you can see in the above figure, p2 contains the address of p (fff2) and p contains the address of number variable (fff4).

1. #include <stdio.h>

2. #include <conio.h>

3. void main(){

4. int number=50;

5. int \*p;//pointer to int

6. int \*\*p2;//pointer to pointer

7. clrscr();

8. p=&number;//stores the address of number variable

9. p2=&p;

10.

11. printf("Address of number variable is %x \n",&number);

12. printf("Address of p variable is %x \n",p);

13. printf("Value of \*p variable is %d \n",\*p);

14. printf("Address of p2 variable is %x \n",p2);

15. printf("Value of \*\*p2 variable is %d \n",\*\*p);

16.

17. getch();

18. }

Output

Address of number variable is fff4

Address of p variable is fff4

Value of \*p variable is 50

Address of p2 variable is fff2

Value of \*\*p variable is 50

Pointer Arithmetic in C

In C pointer holds the address of a value, so there can be arithmetic operations on the pointer variable. Following arithmetic operations are possible on the pointer in C language:

o Increment

o Decrement

o Addition

o Subtraction

o Comparison

Incrementing Pointer in C

Incrementing a pointer is used in the array because it is a contiguous memory location. Moreover, we know the value of the next location.

Increment operation depends on the data type of the pointer variable. The formula of incrementing pointer is given below:

1. new\_address= current\_address + i \* size\_of(data type)

32 bit

For 32-bit int variable, it will increment to 2 bytes.

64 bit

For 64-bit int variable, it will increment to 4 bytes

.

Let's see the example of incrementing pointer variable on 64 bit OS.

1. #include <stdio.h>

2. void main(){

3. int number=50;

4. int \*p;//pointer to int

5. p=&number;//stores the address of number variable

6.

7. printf("Address of p variable is %u \n",p);

8. p=p+1;

9. printf("After increment: Address of p variable is %u \n",p);

10. }

Output

Address of p variable is 3214864300

After increment: Address of p variable is 3214864304

Decrementing Pointer in C

Like increment, we can decrement a pointer variable. The formula of decrementing pointer is given below:

1. new\_address= current\_address - i \* size\_of(data type)

32 bit

For 32-bit int variable, it will decrement to 2 bytes.

64 bit

For 64-bit int variable, it will decrement to 4 bytes.

Let's see the example of decrementing pointer variable on 64 bit OS.

1. #include <stdio.h>

2. void main(){

3. int number=50;

4. int \*p;//pointer to int

5. p=&number;//stores the address of number variable

6.

7. printf("Address of p variable is %u \n",p);

8. p=p-1;

9. printf("After decrement: Address of p variable is %u \n",p);

10. }

Output

Address of p variable is 3214864300

After decrement: Address of p variable is 3214864296

C Pointer Addition

We can add a value to the pointer variable. The formula of adding value to pointer is given below:

1. new\_address= current\_address + (number \* size\_of(data type))

32 bit

For 32 bit int variable, it will add 2 \* number.

64 bit

For 64 bit int variable, it will add 4 \* number.

Let's see the example of adding value to pointer variable on 64 bit OS.

1. #include <stdio.h>

2. void main(){

3. int number=50;

4. int \*p;//pointer to int

5. p=&number;//stores the address of number variable

6.

7. printf("Address of p variable is %u \n",p);

8. p=p+3; //adding 3 to pointer variable

9. printf("After adding 3: Address of p variable is %u \n",p);

10. }

Output

Address of p variable is 3214864300

After adding 3: Address of p variable is 3214864312

As you can see, address of p is 3214864300. But after adding 3 with p variable, it is 3214864312 i.e. 4\*3=12 increment. Since we are using 64 bit OS, it increments 12. But if we were using 32 bit OS, it were incrementing to 6 only i.e. 2\*3=6. As integer value occupies 2 byte memory in 32 bit OS.

C Pointer Subtraction

Like pointer addition, we can subtract a value from the pointer variable. The formula of subtracting value from pointer variable is given below:

1. new\_address= current\_address - (number \* size\_of(data type))

32 bit

For 32 bit int variable, it will subtract 2 \* number.

64 bit

For 64 bit int variable, it will subtract 4 \* number.

Let's see the example of subtracting value from pointer variable on 64 bit OS.

1. #include <stdio.h>

2. void main(){

3. int number=50;

4. int \*p;//pointer to int

5. p=&number;//stores the address of number variable

6.

7. printf("Address of p variable is %u \n",p);

8. p=p-3; //subtracting 3 from pointer variable

9. printf("After subtracting 3: Address of p variable is %u \n",p);

10. }

Output

Address of p variable is 3214864300

After subtracting 3: Address of p variable is 3214864288

You can see after subtracting 3 from pointer variable, it is 12 (4\*3) less than the previous address value.

Dynamic memory allocation in C

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.

1. malloc()

2. calloc()

3. realloc()

4. free()

Before learning the above functions, let's understand the difference between static memory allocation and dynamic memory allocation.

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 the program. memory can be increased while executing the program.

used in the array. used in the linked list.

Now let's have a quick look at the methods used for 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 a 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.

The syntax of malloc() function is given below:

1. ptr=(cast-type\*)malloc(byte-size)

1. #include <stdio.h>

2. #include <stdlib.h>

3. void main(){

4. int n,i,\*ptr,sum=0;

5. printf("Enter number of elements: ");

6. scanf("%d",&n);

7. ptr=(int\*)malloc(n\*sizeof(int)); //memory allocated using calloc

8. if(ptr==NULL)

9. {

10. printf("Sorry! unable to allocate memory");

11. exit(0);

12. }

13. printf("Enter elements of array: ");

14. for(i=0;i<n;++i)

15. {

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

17. sum+=\*(ptr+i);

18. }

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

20. free(ptr);

21. }

Output:

Enter elements of array: 3

Enter elements of array: 10

10

10

Sum=30

calloc() function in C

The calloc() function allocates multiple blocks of requested memory.

It initially initialize all bytes to zero.

It returns NULL if memory is not sufficient.

The syntax of calloc() function is given below:

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

Let's see the example of calloc() function.

1. #include <stdio.h>

2. #include <stdlib.h>

3. void main(){

4. int n,i,\*ptr,sum=0;

5. printf("Enter number of elements: ");

6. scanf("%d",&n);

7. ptr=(int\*)calloc(n,sizeof(int)); //memory allocated using calloc

8. if(ptr==NULL)

9. {

10. printf("Sorry! unable to allocate memory");

11. exit(0);

12. }

13. printf("Enter elements of array: ");

14. for(i=0;i<n;++i)

15. {

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

17. sum+=\*(ptr+i);

18. }

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

20. free(ptr);

21. }

Output:

Enter elements of array: 3

Enter elements of array: 10

10

10

Sum=30

realloc() function in C

If memory is not sufficient for malloc() or calloc(), you can reallocate the memory by realloc() function. In short, it changes the memory size.

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

1. 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.

1. free(ptr)