POINTERS

Pointers to Arrays

Storing elements in a array using pointer

#include <stdio.h>

int main()

{

int i,class[6],sum=0;

printf("Enter 6 numbers:\n");

for(i=0;i<6;++i){

scanf("%d",(class+i)); // (class+i) is equivalent to &class[i]

sum += \*(class+i); // \*(class+i) is equivalent to class[i]

}

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

return 0;

}

Fetching Elements from an array

main()

{

     int a[3] ={1,2,3};

     int \*p;

     printf(“Accessing the element of one dimensional array by pointer variable”);

     p=&a[0];

     for (int count=0;count<=2;count++)

                        {

                                    printf(“%d \n“, \*(p+count));

                        }

 }

POINTERS IN ONE-DIMENSIONAL ARRAY

main()

{

int \*p, sum, i;

int x[5] = {5,9,6,3,7};

i = 0;

p = x; /\* initializing with base address of x \*/

printf("Element Value Address\n\n");

while(i < 5)

{

printf(" x[%d] %d %u\n", i, \*p, p);

sum = sum + \*p; /\* accessing array element \*/

i++, p++; /\* incrementing pointer \*/

}

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

printf("\n &x[0] = %u\n", &x[0]);

printf("\n p = %u\n", p);

}

Output

Element Value Address

x[0] 5 166

x[1] 9 168

x[2] 6 170

x[3] 3 172

x[4] 7 174

Sum = 55

&x[0] = 166

p = 176

Pointer and Functions

Program to swap two number using call by reference.

#include <stdio.h>

void swap(int \*a,int \*b);

int main()

{

int num1=5,num2=10;

swap(&num1,&num2);

printf("Number1 = %d\n",num1);

printf("Number2 = %d",num2);

return 0;

}

void swap(int \*a,int \*b)

{

int temp;

temp=\*a;

\*a=\*b;

\*b=temp;

}

Pointers to Character Strings

Char str[]=”program”

Main()

{

Char \*str=”program”;

//Printf(“%s”, str);

While(\*str!=’\0’)

{

Printf(“%c”, \*str);

Str++;

}

}

Array of pointers to Strings

Printing Address of the Character Array

#include<stdio.h>

int main()

{

int i;

char \*arr[4] = {"C","C++","Java","VBA"};

char \*(\*ptr)[4] = &arr;

for(i=0;i<4;i++)

printf("Address of String %d : %u\n",i+1,(\*ptr)[i]);

return 0;

}

Printing Contents of character array

#include<stdio.h>

int main()

{

int i;

char \*arr[4] = {"C","C++","Java","VBA"};

char \*(\*ptr)[4] = &arr;

for(i=0;i<4;i++)

printf("String %d : %s\n",i+1,(\*ptr)[i]);

return 0;

}

Output :

String 1 = C

String 2 = C++

String 3 = Java

String 4 = VBA

program illustrating pointers to text strings,

#include <stdio.h>

main()

{

static char \*days[] = {"Sunday", "Monday", "Tuesday", "Wednesday", \

"Thursday", "Friday", "Saturday"};

int i;

for( i = 0; i < 6; ++i )

printf( "%s\n", days[i]);

}

**SORTING OF NAMES USING ARRAY OF POINTERS**

#include<stdio.h>

#include<conio.h>

#include<string.h>

#include<stdlib.h>

#define MAX 10

void main()

{

clrscr();

char \*names[MAX],temp[15],\*temp1;

int n,i,j,len;

printf("Enter the no. of names to be sorted");

scanf("%d",&n);

\*names=NULL;

printf("enter the names one by one\n");

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

{

gets(temp);

len=strlen(temp);

\*(names+i)=(char\*)malloc(len+1);

strcpy(\*(names+i),temp);

}

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

for(j=i+1;j<n;j++)

{

if(strcmp(\*(names+i),\*(names+j))>0)

{

temp1=\*(names+i);

\*(names+i)=\*(names+j);

\*(names+j)=temp1;

}

}

printf("\n the sorted list of names:\n");

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

printf("%s\n",\*(names+i));

getch();

}

Pointers to Structures

Pointers can be accessed along with structures. A pointer variable of structure can be created as below:

struct name {

member1;

member2;

.

.

};

struct name \*ptr;

An example to access structure's member through pointer.

#include <stdio.h>

struct name

{

int a;

float b;

};

int main(){

struct name \*ptr,p;

ptr=&p; /\* Referencing pointer to memory address of p \*/

printf("Enter integer: ");

scanf("%d",&(\*ptr).a);

printf("Enter number: ");

scanf("%f",&(\*ptr).b);

printf("Displaying: ");

printf("%d%f",(\*ptr).a,(\*ptr).b);

return 0;

}

Structure pointer member can also be accessed using -> operator.

(\*ptr).a is same as ptr->a

(\*ptr).b is same as ptr->b

Accessing structure member through pointer using dynamic memory allocation

To access structure member using pointers, memory can be allocated dynamically using malloc() function defined under "stdlib.h" header file.

Syntax to use malloc()

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

Example to use structure's member through pointer using malloc() function.

#include <stdio.h>

#include<stdlib.h>

struct name

{

int a;

float b;

char c[30];

};

int main()

{

struct name \*ptr;

int i,n;

printf("Enter n: ");

scanf("%d",&n);

ptr=(struct name\*)malloc(n\*sizeof(struct name));

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

{

printf("Enter string, integer and floating number respectively:\n");

scanf("%s%d%f",&(ptr+i)->c,&(ptr+i)->a,&(ptr+i)->b);

}

printf("Displaying Infromation:\n");

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

printf("%s\t%d\t%.2f\n",(ptr+i)->c,(ptr+i)->a,(ptr+i)->b);

return 0;

}

Output

Enter n: 2

Enter string, integer and floating number respectively:

Programming

2

3.2

Enter string, integer and floating number respectively:

Structure

6

2.3

Displaying Information

Programming 2 3.20

Structure 6 2.30

Pointer Expression

Struct

{

Int count;

Float \*p;

}ptr;

++ptr->count

(++ptr)->count

Ptr++->count

\*ptr->p++

Pointers to Function

#include <stdio.h>

// A normal function with an int parameter

// and void return type

void fun(int a)

{

    printf("Value of a is %d\n", a);

}

int main()

{

    // fun\_ptr is a pointer to function fun()

    void (\*fun\_ptr)(int) = &fun;

 /\* The above line is equivalent of following two

       void (\*fun\_ptr)(int);

       fun\_ptr = &fun;

    \*/

    // Invoking fun() using fun\_ptr

    (\*fun\_ptr)(10);

    return 0;

}

Some interesting facts about function pointers.

1) Unlike normal pointers, a function pointer points to code, not data. Typically a function pointer stores the start of executable code.

2) Unlike normal pointers, we do not allocate de-allocate memory using function pointers.

3) A function’s name can also be used to get functions’ address. For example, in the below program, we have removed address operator ‘&’ in assignment. We have also changed function call by removing \*, the program still works.

4) Like normal pointers, we can have an array of function pointers. Below example in point 5 shows syntax for array of pointers.

5) Function pointer can be used in place of switch case. For example, in below program, user is asked for a choice between 0 and 2 to do different tasks.

**6)**Like normal data pointers, a function pointer can be passed as an argument and can also be returned from a function.

Array of function pointer

|  |
| --- |
| #include <stdio.h>  void add(int a, int b)  {      printf("Addition is %d\n", a+b);  }  void subtract(int a, int b)  {      printf("Subtraction is %d\n", a-b);  }  void multiply(int a, int b)  {      printf("Multiplication is %d\n", a\*b);  }    int main()  {      // fun\_ptr\_arr is an array of function pointers      void (\*fun\_ptr\_arr[])(int, int) = {add, subtract, multiply};      unsigned int ch, a = 15, b = 10;        printf("Enter Choice: 0 for add, 1 for subtract and 2 "              "for multiply\n");      scanf("%d", &ch);        if (ch > 2) return 0;        (\*fun\_ptr\_arr[ch])(a, b);        return 0;  } |

Enter Choice: 0 for add, 1 for subtract and 2 for multiply

2

Multiplication is 150

Passing function pointer as an argument

|  |
| --- |
| #include <stdio.h>  #include <stdbool.h>    bool compare (const void \* a, const void \* b)  {    return ( \*(int\*)a == \*(int\*)b );  }    int search(void \*arr, int arr\_size, int ele\_size, void \*x,             bool compare (const void \* , const void \*))  {      char \*ptr = (char \*)arr;        int i;      for (i=0; i<arr\_size; i++)          if (compare(ptr + i\*ele\_size, x))             return i;        // If element not found      return -1;  }    int main()  {      int arr[] = {2, 5, 7, 90, 70};      int n = sizeof(arr)/sizeof(arr[0]);      int x = 7;      printf ("Returned index is %d ", search(arr, n,sizeof(int), &x, compare));      return 0;  } |

Output:

Returned index is 2

Passing a Function Pointer as a Parameter

lets look at using a function pointer to execute different functions based on input.

|  |  |
| --- | --- |
|  | #include <stdio.h>    // function prototypes  int add(int x, int y);  int subtract(int x, int y);  int domath(int (\*mathop)(int, int), int x, int y);    // add x + y  int add(int x, int y)  {    return x + y;  }    // subtract x - y  int subtract(int x, int y)  {    return x - y;  }    // run the function pointer with inputs  int domath(int (\*mathop)(int, int), int x, int y)  {    return (\*mathop)(x, y);  }    // calling from main  int main()  {    // call math function with add    int a = domath(add, 10, 2);    printf("Add gives: %d\n", a);      // call math function with subtract    int b = domath(subtract, 10, 2);    printf("Subtract gives: %d\n", b);  } |

Let’s break this down.

We have two functions with the same signature int function(int, int), add and subtract. Both return an integer and both take two integers as parameters.

On line 6 we have int domath(int (\*mathop)(int, int), int x, int y) The first parameter int (\*mathop)(int, int) is a pointer to a function that takes two integers as input and returns an integer. We have seen this before, the syntax is no different here. The last two parameters x and y are just integer inputs into the domath function. So the domath function is takes a function pointer and two integers as parameters.

On lines 19-21 the domath function executes the function pointer passed with the x and y integers passed. This could also have been done as mathop(x, y);.

Lines 27 and 31 are somewhat new. We are calling the domath function and we are passing in the function names. Function names are the address-of the function and can be used in place of function pointers.

The main function calls domath twice, once for add and once for subtract, printing out the results.