Double pointers

#include <stdio.h>

int main()

{

int i = 5,j=6,k=7;

int \*ip1 = &i;

printf("Address of i = %p\n",&i);

printf("Address of j = %p\n",&j);

printf("Address of k = %p\n",&k);

printf("value of ip1 = %d\n",\*ip1);

int \*ip2 = &j;

printf("value of ip2 = %d\n",\*ip2);

int \*\*ipp = &ip1;

// \*ipp=ip2;

\*ipp = &k;

printf("\*ipp = %p\n",\*ipp);

printf("\*\*ipp = %d\n", \*\*ipp);

return 0;

}

#include<stdio.h>

#include<stdlib.h>

int main()

{

int num = 20;

int \*p = &num;

int \*\*dp = &p;

printf("001 num = %d\n",\*p);

printf("Address of num = %p\n",&num);

printf("002 num = %d\n", \*\*dp);

return 0;

}

**1. Reverse a String**

Write a function void reverseString(char \*str) that takes a pointer to a string and reverses the string in place.

#include<stdio.h>

void reverse(char \*str);

int main()

{

char str[100];

printf("enter a string :");

scanf("%s",str);

reverse(str);

printf("reversed string is %s ",str);

return 0;

}

void reverse(char \*str){

char \*start = str;

char \* end = str;

char temp;

while(\*end != '\0'){

end++;

}

end--;

while(start < end){

temp = \*start;

\*start = \*end;

\*end = temp;

start++;

end--;

}

}

**2. Concatenate Two Strings**

Implement a function void concatenateStrings(char \*dest, const char \*src) that appends the source string to the destination string using pointers.

#include<stdio.h>

void concantenate(char \*dest, char \*src);

int main()

{

char dest[100],src[100];

printf("enter the destinaton string: ");

scanf("%s",dest);

printf("enter the source string : ");

scanf("%s",src);

concantenate(dest, src);

printf("concatenated string is %s ",dest);

return 0;

}

void concantenate(char \*dest, char \*src){

while(\*dest!='\0'){

dest++;

}

while(\*src !='\0'){

\*dest = \*src;

dest ++;

src++;

}

\*dest = '\0';

}

**3. String Length**

Create a function int stringLength(const char \*str) that calculates and returns the length of a string using pointers.

#include<stdio.h>

int stringlength( const char \*str);

int main()

{

char str[100];

printf("enter a string : ");

scanf("%s",str);

int length = stringlength(str);

printf("length of the string is %d",length);

return 0;

}

int stringlength(char const \*str){

const char \*ptr = str;

while(\*ptr != '\0'){

ptr++;

}

return ptr-str;

}

**4. Compare Two Strings**

Write a function int compareStrings(const char \*str1, const char \*str2) that compares two strings lexicographically and returns 0 if they are equal, a positive number if str1 is greater, or a negative number if str2 is greater.

#include <stdio.h>

int compareStrings(const char \*str1, const char \*str2) {

while (\*str1 && (\*str1 == \*str2)) {

str1++;

str2++;

}

return \*(unsigned char \*)str1 - \*(unsigned char \*)str2;

}

int main() {

char str1[100];

char str2[100];

printf("Enter the first string: ");

scanf("%99s", str1);

printf("Enter the second string: ");

scanf("%99s", str2);

int result = compareStrings(str1, str2);

if (result == 0) {

printf("The strings are equal.\n");

} else if (result > 0) {

printf("The first string is greater than the second string.\n");

} else {

printf("The first string is less than the second string.\n");

}

return 0;

}

**5. Find Substring**

Implement char\* findSubstring(const char \*str, const char \*sub) that returns a pointer to the first occurrence of the substring sub in the string str, or NULL if the substring is not found.

#include <stdio.h>

char\* findSubstring(const char \*str, const char \*sub) {

if (!str || !sub) {

return NULL;

}

while (\*str) {

const char \*start = str;

const char \*pattern = sub;

while (\*str && \*pattern && \*str == \*pattern) {

str++;

pattern++;

}

if (!\*pattern) {

return (char \*)start;

}

str = start + 1;

}

return NULL;

}

int main() {

const char \*mainText = "Hello world! Welcome to the world of programming.";

const char \*substring = "world";

char \*result = findSubstring(mainText, substring);

if (result) {

printf("Substring found at position: %ld\n", result - mainText);

} else {

printf("Substring not found.\n");

}

return 0;

}

**6. Replace Character in String**

Write a function void replaceChar(char \*str, char oldChar, char newChar) that replaces all occurrences of oldChar with newChar in the given string.

#include <stdio.h>

void replaceChar(char \*str, char oldChar, char newChar) {

while (\*str != '\0') {

if (\*str == oldChar) {

\*str = newChar;

}

str++;

}

}

int main() {

char str[100];

char oldChar, newChar;

printf("Enter a string: ");

scanf("%99s", str);

printf("Enter the character to replace: ");

scanf(" %c", &oldChar);

printf("Enter the new character: ");

scanf(" %c", &newChar);

replaceChar(str, oldChar, newChar);

printf("Modified string: %s\n", str);

return 0;

}

**7. Copy String**

Create a function void copyString(char \*dest, const char \*src) that copies the content of the source string src to the destination string dest.

#include <stdio.h>

void copyString(char \*dest, const char \*src) {

while (\*src != '\0') {

\*dest = \*src;

dest++;

src++;

}

\*dest = '\0';

}

int main() {

char src[100];

char dest[100];

printf("Enter the source string: ");

scanf("%s", src);

copyString(dest, src);

printf("Copied string: %s\n", dest);

return 0;

}

**8. Count Vowels in a String**

Implement int countVowels(const char \*str) that counts and returns the number of vowels in a given string.

#include <stdio.h>

int countVowels(const char \*str) {

int count = 0;

char ch;

while (\*str != '\0') {

ch = \*str;

if (ch == 'A' || ch == 'E' || ch == 'I' || ch == 'O' || ch == 'U' ||

ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u') {

count++;

}

str++;

}

return count;

}

int main() {

char str[100];

printf("Enter a string: ");

scanf("%99s", str);

int result = countVowels(str);

printf("The number of vowels in the string is: %d\n", result);

return 0;

}

**9. Check Palindrome**

Write a function int isPalindrome(const char \*str) that checks if a given string is a palindrome and returns 1 if true, otherwise 0.

#include <stdio.h>

#include <string.h>

int isPalindrome(const char \*str) {

int length = strlen(str);

const char \*start = str;

const char \*end = str + length - 1;

while (start < end) {

if (\*start != \*end) {

return 0;

}

start++;

end--;

}

return 1;

}

int main() {

char str[100];

printf("Enter a string: ");

scanf("%99s", str);

int result = isPalindrome(str);

if (result) {

printf("The string is a palindrome.\n");

} else {

printf("The string is not a palindrome.\n");

}

return 0;

}

**10. Tokenize String**

Create a function void tokenizeString(char \*str, const char \*delim, void (\*processToken)(const char \*)) that tokenizes the string str using delimiters in delim, and for each token, calls processToken.

#include <stdio.h>

#include <string.h>

void tokenizeString(char \*str, const char \*delim, void (\*processToken)(const char \*)) {

char \*token = strtok(str, delim);

while (token != NULL) {

processToken(token);

token = strtok(NULL, delim);

}

}

void printToken(const char \*token) {

printf("Token: %s\n", token);

}

int main() {

char str[100];

char delim[10];

printf("Enter the string to tokenize: ");

scanf("%99[^\n]%\*c", str);

printf("Enter the delimiters: ");

scanf("%9s", delim);

tokenizeString(str, delim, printToken);

return 0;

}

**1. Allocate and Free Integer Array**

Write a program that dynamically allocates memory for an array of integers, fills it with values from 1 to n, and then frees the allocated memory.

#include<stdio.h>

#include<stdlib.h>

int main()

{

int n;

printf("enter the no of elemnts: ");

scanf("%d",&n);

int \*arr = (int \*)malloc(sizeof(int) );

if(arr==NULL){

printf("allocation failed");

return 1;

}

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

arr[i]=i+1;

}

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

printf("%d ",arr[i]);

}

printf("\n");

free(arr);

}

**2. Dynamic String Input**

Implement a function that dynamically allocates memory for a string, reads a string input from the user, and then prints the string. Free the memory after use.

#include <stdio.h>

#include <stdlib.h>

char\* getInputString() {

char \*str;

int length;

printf("Enter the length of the string: ");

scanf("%d", &length);

str = (char \*)malloc((length + 1) \* sizeof(char));

if (str == NULL) {

printf("Memory allocation failed!\n");

exit(1);

}

printf("Enter the string: ");

scanf(" ");

for (int i = 0; i < length; i++) {

str[i] = getchar();

}

str[length] = '\0';

return str;

}

int main() {

char \*inputString = getInputString();

printf("You entered: %s\n", inputString);

free(inputString);

return 0;

}

**3. Resize an Array**

Write a program that dynamically allocates memory for an array of n integers, fills it with values, resizes the array to 2n using realloc(), and fills the new elements with values.

#include <stdio.h>

#include <stdlib.h>

int main() {

int \*array;

int n;

printf("Enter the number of elements: ");

scanf("%d", &n);

array = (int \*)malloc(n \* sizeof(int));

if (array == NULL) {

printf("Memory allocation failed!\n");

return 1;

}

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

array[i] = i + 1;

}

printf("Initial array values: ");

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

printf("%d ", array[i]);

}

printf("\n");

array = (int \*)realloc(array, 2 \* n \* sizeof(int));

if (array == NULL) {

printf("Memory reallocation failed!\n");

return 1;

}

for (int i = n; i < 2 \* n; i++) {

array[i] = i + 1;

}

printf("Resized array values: ");

for (int i = 0; i < 2 \* n; i++) {

printf("%d ", array[i]);

}

printf("\n");

free(array);

return 0;

}

**4. Matrix Allocation**

Create a function that dynamically allocates memory for a 2D array (matrix) of size m x n, fills it with values, and then deallocates the memory.

#include <stdio.h>

#include <stdlib.h>

// Function to dynamically allocate memory for a 2D array

int\* allocateMatrix(int rows, int cols) {

return (int\*)malloc(rows \* cols \* sizeof(int));

}

void fillMatrix(int\* matrix, int rows, int cols) {

printf("Enter elements for the %dx%d matrix:\n", rows, cols);

for (int i = 0; i < rows; i++) {

for (int j = 0; j < cols; j++) {

printf("Element [%d][%d]: ", i, j);

scanf("%d", (matrix + i \* cols + j));

}

}

}

// Function to print the matrix

void printMatrix(int\* matrix, int rows, int cols) {

printf("The matrix is:\n");

for (int i = 0; i < rows; i++) {

for (int j = 0; j < cols; j++) {

printf("%d ", \*(matrix + i \* cols + j));

}

printf("\n");

}

}

int main() {

int rows, cols;

printf("Enter the number of rows: ");

scanf("%d", &rows);

printf("Enter the number of columns: ");

scanf("%d", &cols);

int\* matrix = allocateMatrix(rows, cols);

if (!matrix) {

printf("Memory allocation failed.\n");

return 1;

}

fillMatrix(matrix, rows, cols);

printMatrix(matrix, rows, cols);

free(matrix);

printf("Memory deallocated successfully.\n");

return 0;

}

**5. String Concatenation with Dynamic Memory**

Implement a function that takes two strings, dynamically allocates memory to concatenate them, and returns the new concatenated string. Ensure to free the memory after use.

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

char\* concatenateStrings(const char \*str1, const char \*str2) {

int len1 = strlen(str1);

int len2 = strlen(str2);

int totalLength = len1 + len2 + 1;

char \*result = (char \*)malloc(totalLength \* sizeof(char));

if (result == NULL) {

printf("Memory allocation failed!\n");

exit(1);

}

strcpy(result, str1);

strcat(result, str2);

return result;

}

int main() {

char str1[100];

char str2[100];

printf("Enter the first string: ");

scanf("%99s", str1);

printf("Enter the second string: ");

scanf("%99s", str2);

char \*concatenatedString = concatenateStrings(str1, str2);

printf("Concatenated string: %s\n", concatenatedString);

free(concatenatedString);

return 0;

}

**6. Dynamic Memory for Structure**

Define a struct for a student with fields like name, age, and grade. Write a program that dynamically allocates memory for a student, fills in the details, and then frees the memory.

#include <string.h>

typedef struct {

char name[50];

int age;

float grade;

} Student;

Student\* createStudent(const char \*name, int age, float grade) {

Student \*student = (Student \*)malloc(sizeof(Student));

if (student == NULL) {

printf("Memory allocation failed!\n");

exit(1);

}

strncpy(student->name, name, sizeof(student->name) - 1);

student->name[sizeof(student->name) - 1] = '\0';

student->age = age;

student->grade = grade;

return student;

}

void displayStudent(const Student \*student) {

if (student != NULL) {

printf("Name: %s\n", student->name);

printf("Age: %d\n", student->age);

printf("Grade: %.2f\n", student->grade);

}

}

void freeStudent(Student \*student) {

free(student);

}

int main() {

char name[50];

int age;

float grade;

printf("Enter student name: ");

scanf("%49s", name);

printf("Enter student age: ");

scanf("%d", &age);

printf("Enter student grade: ");

scanf("%f", &grade);

Student \*student = createStudent(name, age, grade);

displayStudent(student);

freeStudent(student);

return 0;

}

**7. Dynamic Array of Pointers**

Write a program that dynamically allocates memory for an array of pointers to integers, fills each integer with values, and then frees all the allocated memory.

#include <stdio.h>

#include <stdlib.h>

int main() {

int n;

printf("Enter the number of elements: ");

scanf("%d", &n);

int \*array = (int \*)malloc(sizeof(int));

if (array == NULL) {

printf("Memory allocation failed!\n");

return 1;

}

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

array[i] = i + 1;

}

printf("Array values:\n");

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

printf("%d ", array[i]);

}

printf("\n");

free(array);

return 0;

}

**8. Dynamic Memory for Multidimensional Arrays**

Create a program that dynamically allocates memory for a 3D array of integers, fills it with values, and deallocates the memory.

#include <stdio.h>

#include <stdlib.h>

int main() {

int \*array;

int x, y, z;

printf("Enter the dimensions of the 3D array (x y z): ");

scanf("%d %d %d", &x, &y, &z);

array = (int \*)malloc(x \* y \* z \* sizeof(int));

if (array == NULL) {

printf("Memory allocation failed!\n");

return 1;

}

for (int i = 0; i < x; i++) {

for (int j = 0; j < y; j++) {

for (int k = 0; k < z; k++) {

array[(i \* y \* z) + (j \* z) + k] = (i \* y \* z) + (j \* z) + k;

}

}

}

printf("3D array values:\n");

for (int i = 0; i < x; i++) {

for (int j = 0; j < y; j++) {

for (int k = 0; k < z; k++) {

printf("array[%d][%d][%d] = %d\n", i, j, k, array[(i \* y \* z) + (j \* z) + k]);

}

}

}

free(array);

return 0;

}

Function pointer

#include <stdio.h>

#include <stdlib.h>

int add(int,int);

int main()

{

int (\*fptr)(int,int) ;//declaration of function pointer

//initialization

fptr = &add;

//call tha function using functon pointer

printf("%d",fptr(10,5));

return 0;

}

int add(int a,int b){

return a+b;

}

**Double Pointers**

**1. Swap Two Numbers Using Double Pointers**

Write a function void swap(int \*\*a, int \*\*b) that swaps the values of two integer pointers using double pointers.

#include<stdio.h>

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

int main()

{

int x=5;

int y= 6;

int \*ptr1 = &x;

int \*ptr2 = &y;

printf("ptr1 pointa to %d amd ptr2 points to %d\n",\*ptr1,\*ptr2);

swap(&ptr1, &ptr2);

printf("after swapping ptr1 points to %d and ptr2 points to %d",\*ptr1,\*ptr2);

}

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

int \*temp = \*a;

\*a= \*b;

\*b = temp;

}

**2. Dynamic Memory Allocation Using Double Pointer**

Implement a function void allocateArray(int \*\*arr, int size) that dynamically allocates memory for an array of integers using a double pointer.

#include <stdio.h>

#include <stdlib.h>

void allocateArray(int \*\*arr, int size) {

\*arr = (int \*)malloc(size \* sizeof(int));

if (\*arr == NULL) {

printf("Memory allocation failed!\n");

exit(1);

}

}

int main() {

int \*array;

int size;

printf("Enter the size of the array: ");

scanf("%d", &size);

allocateArray(&array, size);

for (int i = 0; i < size; i++) {

array[i] = i + 1;

}

printf("Array values: ");

for (int i = 0; i < size; i++) {

printf("%d ", array[i]);

}

printf("\n");

free(array);

return 0;

}

**3. Modify a String Using Double Pointer**

Write a function void modifyString(char \*\*str) that takes a double pointer to a string, dynamically allocates a new string, assigns it to the pointer, and modifies the original string.

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

void modifyString(char \*\*str) {

char \*newStr = (char \*)malloc(100 \* sizeof(char));

if (newStr == NULL) {

printf("Memory allocation failed!\n");

exit(1);

}

strcpy(newStr, "This is the new modified string.");

if (\*str != NULL) {

free(\*str);

}

\*str = newStr;

}

int main() {

char \*str = NULL;

modifyString(&str);

printf("Modified string: %s\n", str);

free(str);

return 0;

}

**4. Pointer to Pointer Example**

Create a simple program that demonstrates how to use a pointer to a pointer to access and modify the value of an integer.

#include <stdio.h>

void modifyValue(int \*\*ptr\_to\_ptr) {

\*\*ptr\_to\_ptr = 20;

}

int main() {

int value = 10;

int \*ptr = &value;

int \*\*ptr\_to\_ptr = &ptr;

printf("Initial value: %d\n", value);

modifyValue(ptr\_to\_ptr);

printf("Modified value: %d\n", value);

return 0;

}

**5. 2D Array Using Double Pointer**

Write a function int\*\* create2DArray(int rows, int cols) that dynamically allocates memory for a 2D array of integers using a double pointer and returns the pointer to the array.

#include <stdio.h>

#include <stdlib.h>

int\*\* create2DArray(int rows, int cols) {

int \*\*array = (int \*\*)malloc(rows \* sizeof(int \*));

if (array == NULL) {

printf("Memory allocation failed!\n");

exit(1);

}

for (int i = 0; i < rows; i++) {

array[i] = (int \*)malloc(cols \* sizeof(int));

if (array[i] == NULL) {

printf("Memory allocation failed!\n");

exit(1);

}

}

return array;

}

void fill2DArray(int \*\*array, int rows, int cols) {

for (int i = 0; i < rows; i++) {

for (int j = 0; j < cols; j++) {

array[i][j] = i \* cols + j;

}

}

}

void print2DArray(int \*\*array, int rows, int cols) {

for (int i = 0; i < rows; i++) {

for (int j = 0; j < cols; j++) {

printf("%d ", array[i][j]);

}

printf("\n");

}

}

void deallocate2DArray(int \*\*array, int rows) {

for (int i = 0; i < rows; i++) {

free(array[i]);

}

free(array);

}

int main() {

int rows, cols;

printf("Enter the number of rows: ");

scanf("%d", &rows);

printf("Enter the number of columns: ");

scanf("%d", &cols);

int \*\*array = create2DArray(rows, cols);

fill2DArray(array, rows, cols);

printf("2D array values:\n");

print2DArray(array, rows, cols);

deallocate2DArray(array, rows);

return 0;

}

**6. Freeing 2D Array Using Double Pointer**

Implement a function void free2DArray(int \*\*arr, int rows) that deallocates the memory allocated for a 2D array using a double pointer.

#include <stdio.h>

#include <stdlib.h>

void free2DArray(int \*\*arr, int rows) {

for (int i = 0; i < rows; i++) {

free(arr[i]);

}

free(arr);

}

int main() {

int rows = 3;

int cols = 4;

int \*\*array = (int \*\*)malloc(rows \* sizeof(int \*));

for (int i = 0; i < rows; i++) {

array[i] = (int \*)malloc(cols \* sizeof(int));

}

for (int i = 0; i < rows; i++) {

for (int j = 0; j < cols; j++) {

array[i][j] = i \* cols + j;

}

}

printf("2D array values:\n");

for (int i = 0; i < rows; i++) {

for (int j = 0; j < cols; j++) {

printf("%d ", array[i][j]);

}

printf("\n");

}

free2DArray(array, rows);

return 0;

}

**7. Pass a Double Pointer to a Function**

Write a function void setPointer(int \*\*ptr) that sets the pointer passed to it to point to a dynamically allocated integer.

#include <stdio.h>

#include <stdlib.h>

void setPointer(int \*\*ptr) {

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

if (\*ptr == NULL) {

printf("Memory allocation failed!\n");

exit(1);

}

\*\*ptr = 42;

}

int main() {

int \*ptr = NULL;

setPointer(&ptr);

printf("Value pointed to by ptr: %d\n", \*ptr);

free(ptr);

return 0;

}

**8. Dynamic Array of Strings**

Create a function void allocateStringArray(char \*\*\*arr, int n) that dynamically allocates memory for an array of n strings using a double pointer.

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

void allocateStringArray(char \*\*\*arr, int n) {

\*arr = (char \*\*)malloc(n \* sizeof(char \*));

if (\*arr == NULL) {

printf("Memory allocation failed!\n");

exit(1);

}

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

(\*arr)[i] = (char \*)malloc(100 \* sizeof(char)); // Example: Allocate memory for 100 characters

if ((\*arr)[i] == NULL) {

printf("Memory allocation failed!\n");

exit(1);

}

}

}

void freeStringArray(char \*\*arr, int n) {

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

free(arr[i]);

}

free(arr);

}

int main() {

char \*\*stringArray;

int n;

printf("Enter the number of strings: ");

scanf("%d", &n);

allocateStringArray(&stringArray, n);

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

snprintf(stringArray[i], 100, "String %d", i + 1); // Example: Fill with "String 1", "String 2", etc.

}

printf("Array of strings:\n");

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

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

}

freeStringArray(stringArray, n);

return 0;

}

**9. String Array Manipulation Using Double Pointer**

Implement a function void modifyStringArray(char \*\*arr, int n) that modifies each string in an array of strings using a double pointer.

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

void modifyStringArray(char \*\*arr, int n) {

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

strcat(arr[i], " - modified");

}

}

int main() {

int n;

char \*\*stringArray;

printf("Enter the number of strings: ");

scanf("%d", &n);

stringArray = (char \*\*)malloc(n \* sizeof(char \*));

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

stringArray[i] = (char \*)malloc(100 \* sizeof(char));

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

scanf("%s", stringArray[i]);

}

modifyStringArray(stringArray, n);

printf("Modified array of strings:\n");

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

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

}

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

free(stringArray[i]);

}

free(stringArray);

return 0;

}

**Function Pointers**

**1. Basic Function Pointer Declaration**

Write a program that declares a function pointer for a function int add(int, int) and uses it to call the function and print the result.

#include <stdio.h>

#include <stdlib.h>

int add(int,int);

int main()

{

int (\*fptr)(int,int) ;//declaration of function pointer

//initialization

fptr = &add;

//call tha function using functon pointer

printf("%d",fptr(10,5));

return 0;

}

int add(int a,int b){

return a+b;

}

**2. Function Pointer as Argument**

Implement a function void performOperation(int (\*operation)(int, int), int a, int b) that takes a function pointer as an argument and applies it to two integers, printing the result.

#include <stdio.h>

void performOperation(int (\*operation)(int, int), int a, int b) {

int result = operation(a, b);

printf("Result: %d\n", result);

}

int add(int a, int b) {

return a + b;

}

int multiply(int a, int b) {

return a \* b;

}

int main() {

int a, b;

printf("Enter two integers:\n");

scanf("%d %d", &a, &b);

printf("Performing addition:\n");

performOperation(add, a, b);

printf("Performing multiplication:\n");

performOperation(multiply, a, b);

return 0;

}

**3. Function Pointer Returning Pointer**

Write a program with a function int\* max(int \*a, int \*b) that returns a pointer to the larger of two integers, and use a function pointer to call this function.

#include <stdio.h>

int\* max(int \*a, int \*b) {

if (\*a > \*b) {

return a;

} else {

return b;

}

}

int main() {

int x = 5;

int y = 10;

int \*result;

int\* (\*funcPtr)(int \*, int \*);

funcPtr = max;

result = funcPtr(&x, &y);

printf("The larger value is: %d\n", \*result);

return 0;

}

**4. Function Pointer with Different Functions**

Create a program that defines two functions int add(int, int) and int multiply(int, int) and uses a function pointer to dynamically switch between these functions based on user input.

#include <stdio.h>

int add(int a, int b) {

return a + b;

}

int multiply(int a, int b) {

return a \* b;

}

int main() {

int a, b, choice;

int (\*operation)(int, int);

printf("Enter two integers:\n");

scanf("%d %d", &a, &b);

printf("Choose an operation:\n");

printf("1. Add\n");

printf("2. Multiply\n");

printf("Enter your choice: ");

scanf("%d", &choice);

if (choice == 1) {

operation = add;

} else if (choice == 2) {

operation = multiply;

} else {

printf("Invalid choice!\n");

return 1;

}

int result = operation(a, b);

printf("Result: %d\n", result);

return 0;

}

**5. Array of Function Pointers**

Implement a program that creates an array of function pointers for basic arithmetic operations (addition, subtraction, multiplication, division) and allows the user to select and execute one operation.

#include <stdio.h>

int add(int a, int b) {

return a + b;

}

int subtract(int a, int b) {

return a - b;

}

int multiply(int a, int b) {

return a \* b;

}

int divide(int a, int b) {

if (b != 0) {

return a / b;

} else {

printf("Error: Division by zero!\n");

return 0;

}

}

int main() {

int a, b, choice;

int result;

int (\*operations[4])(int, int) = { add, subtract, multiply, divide };

printf("Enter two integers: ");

scanf("%d %d", &a, &b);

printf("Choose an operation:\n");

printf("0. Add\n");

printf("1. Subtract\n");

printf("2. Multiply\n");

printf("3. Divide\n");

printf("Enter your choice: ");

scanf("%d", &choice);

if (choice >= 0 && choice < 4) {

result = operations[choice](a, b);

printf("Result: %d\n", result);

} else {

printf("Invalid choice!\n");

}

return 0;

}

**6. Using Function Pointers for Sorting**

Write a function void sort(int \*arr, int size, int (\*compare)(int, int)) that uses a function pointer to compare elements, allowing for both ascending and descending order sorting.

#include <stdio.h>

int ascending(int a, int b) {

return a - b;

}

int descending(int a, int b) {

return b - a;

}

void sort(int \*arr, int size, int (\*compare)(int, int)) {

for (int i = 0; i < size - 1; i++) {

for (int j = 0; j < size - 1 - i; j++) {

if (compare(arr[j], arr[j + 1]) > 0) {

int temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

}

}

}

}

int main() {

int arr[] = {5, 3, 8, 6, 2, 7, 4, 1};

int size = sizeof(arr) / sizeof(arr[0]);

int choice;

printf("Choose sorting order:\n");

printf("1. Ascending\n");

printf("2. Descending\n");

printf("Enter your choice: ");

scanf("%d", &choice);

int (\*compare)(int, int);

if (choice == 1) {

compare = ascending;

} else if (choice == 2) {

compare = descending;

} else {

printf("Invalid choice!\n");

return 1;

}

sort(arr, size, compare);

printf("Sorted array:\n");

for (int i = 0; i < size; i++) {

printf("%d ", arr[i]);

}

printf("\n");

return 0;

}

**7. Callback Function**

Create a program with a function void execute(int x, int (\*callback)(int)) that applies a callback function to an integer and prints the result. Demonstrate with multiple callback functions (e.g., square, cube).

#include <stdio.h>

int square(int x) {

return x \* x;

}

int cube(int x) {

return x \* x \* x;

}

void execute(int x, int (\*callback)(int)) {

int result = callback(x);

printf("Result: %d\n", result);

}

int main() {

int x;

int choice;

printf("Enter an integer: ");

scanf("%d", &x);

printf("Choose an operation:\n");

printf("1. Square\n");

printf("2. Cube\n");

printf("Enter your choice: ");

scanf("%d", &choice);

int (\*callback)(int);

if (choice == 1) {

callback = square;

} else if (choice == 2) {

callback = cube;

} else {

printf("Invalid choice!\n");

return 1;

}

execute(x, callback);

return 0;

}

**8. Menu System Using Function Pointers**

Implement a simple menu system where each menu option corresponds to a different function, and a function pointer array is used to call the selected function based on user input.

#include <stdio.h>

#include <stdlib.h>

void option1() {

printf("You selected option 1.\n");

}

void option2() {

printf("You selected option 2.\n");

}

void option3() {

printf("You selected option 3.\n");

}

void exitProgram() {

printf("Exiting the program.\n");

exit(0);

}

int main() {

int choice;

void (\*menu[4])() = { option1, option2, option3, exitProgram };

while (1) {

printf("\nMenu:\n");

printf("1. Option 1\n");

printf("2. Option 2\n");

printf("3. Option 3\n");

printf("4. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

if (choice >= 1 && choice <= 4) {

menu[choice - 1]();

} else {

printf("Invalid choice! Please try again.\n");

}

}

return 0;

}

**9. Dynamic Function Selection**

Write a program where the user inputs an operation symbol (+, -, \*, /) and the program uses a function pointer to call the corresponding function.

#include <stdio.h>

#include <stdlib.h>

int add(int a, int b) {

return a + b;

}

int subtract(int a, int b) {

return a - b;

}

int multiply(int a, int b) {

return a \* b;

}

int divide(int a, int b) {

if (b != 0) {

return a / b;

} else {

printf("Error: Division by zero!\n");

exit(1);

}

}

int main() {

int a, b;

char operation;

int (\*operationFunc)(int, int);

printf("Enter two integers: ");

scanf("%d %d", &a, &b);

printf("Enter an operation (+, -, \*, /): ");

scanf(" %c", &operation);

switch (operation) {

case '+':

operationFunc = add;

break;

case '-':

operationFunc = subtract;

break;

case '\*':

operationFunc = multiply;

break;

case '/':

operationFunc = divide;

break;

default:

printf("Invalid operation!\n");

return 1;

}

int result = operationFunc(a, b);

printf("Result: %d\n", result);

return 0;

}

**10. State Machine with Function Pointers**

Design a simple state machine where each state is represented by a function, and transitions are handled using function pointers. For example, implement a traffic light system with states like Red, Green, and Yellow.

#include <stdio.h>

#include <stdlib.h>

typedef enum {

RED,

GREEN,

YELLOW,

NUM\_STATES

} State;

void redState();

void greenState();

void yellowState();

void (\*stateFunctions[NUM\_STATES])() = { redState, greenState, yellowState };

State currentState;

void nextState() {

stateFunctions[currentState]();

currentState = (currentState + 1) % NUM\_STATES;

}

void redState() {

printf("RED light - Stop!\n");

}

void greenState() {

printf("GREEN light - Go!\n");

}

void yellowState() {

printf("YELLOW light - Slow down!\n");

}

int main() {

currentState = RED;

for (int i = 0; i < 10; i++) {

nextState();

}

return 0;

}