Day9_assignment_Abhirami

Problem 1: Array Element Access

Write a program in C that demonstrates the use of a pointer to a const array of integers. The program should do the following:

1. Define an integer array with fixed values (e.g., {1, 2, 3, 4, 5}).

2. Create a pointer to this array that uses the const qualifier to ensure that the elements cannot be modified through the pointer.

3. Implement a function printArray(const int *arr, int size) to print the elements of the array using the const pointer.

4. Attempt to modify an element of the array through the pointer (this should produce a compilation error, demonstrating the behavior of const).

Requirements:

a. Use a pointer of type const int* to access the array.

b. The function should not modify the array elements.

```
#include<stdio.h>
void pointArray(const int*arr,int size);
int main() {
   const int arr[]={1,2,3,4,5};
   printf("array elements are: ");
 pointArray(&arr[0],5);
   // ptr[0]=10;..uncommenting this cause an error

    return 0;
}

void pointArray(const int*arr,int size){
    for(int i=0;i<size;i++) {
        printf("%d",arr[i]);
    }

}
```

Problem 2: Protecting a Value

Write a program in C that demonstrates the use of a pointer to a const integer and a const pointer to an integer. The program should:

1. Define an integer variable and initialize it with a value (e.g., int value = 10;).

2. Create a pointer to a const integer and demonstrate that the value cannot be modified through the pointer.

3. Create a const pointer to the integer and demonstrate that the pointer itself cannot be changed to point to another variable.

4. Print the value of the integer and the pointer address in each case.

Requirements:

a. Use the type qualifiers const int* and int* const appropriately.

b. Attempt to modify the value or the pointer in an invalid way to show how the compiler enforces the constraints.

```
#include <stdio.h>

int main() {
      int value = 10;
   int anotherValue = 20;

     const int *ptrToConst = &value;
   printf("Pointer to const integer:\n");
   printf("Value pointed to by ptrToConst: %d\n", *ptrToConst);

   // Attempt to modify the value through the pointer (uncomment to see compilation error)
   // *ptrToConst = 15; // Error: cannot modify a const value through the pointer

    int *const constPtr = &value;
   printf("\nConst pointer to an integer:\n");
   printf("Value pointed to by constPtr: %d\n", *constPtr);

      *constPtr = 30; // Valid: the pointer itself is constant, but the value it points to is not
```

```
    printf("Modified value through constPtr: %d\n", *constPtr);

    // Attempt to change the pointer itself (uncomment to see compilation error)
    // constPtr = &anotherValue; // Error: cannot change the address stored in a const pointer

      printf("\nMemory addresses:\n");
    printf("Address stored in ptrToConst: %p\n", (void *)ptrToConst);
    printf("Address stored in constPtr: %p\n", (void *)constPtr);

    return 0;
}
```

/*
Problem: Universal Data Printer
You are tasked with creating a universal data printing function in C that can handle different types of data (int, float, and char*). The function should use void pointers to accept any type of data and print it appropriately based on a provided type specifier.

Specifications
Implement a function print_data with the following signature:

```
void print_data(void* data, char type);
```

Parameters:

data: A void* pointer that points to the data to be printed.

type: A character indicating the type of data:
- 'i' for int
- 'f' for float
- 's' for char* (string)

Behavior:
- If type is 'i', interpret data as a pointer to int and print the integer.
- If type is 'f', interpret data as a pointer to float and print the floating-point value.
- If type is 's', interpret data as a pointer to a char* and print the string.

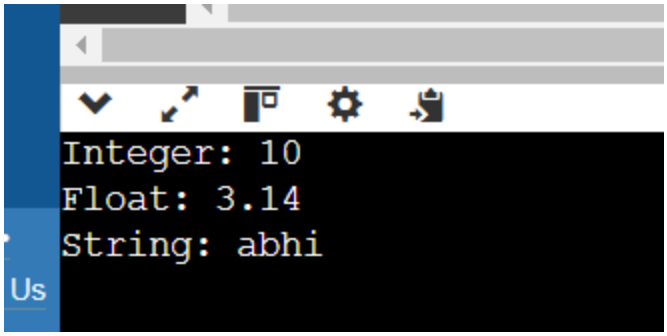
In the main function:
- Declare variables of types int, float, and char*.
- Call print_data with these variables using the appropriate type specifier.

Example output:
Input data: 42 (int), 3.14 (float), "Hello, world!" (string)
Output:

Integer: 42
Float: 3.14
String: Hello, world!

Constraints
1. Use void* to handle the input data.
2. Ensure that typecasting from void* to the correct type is performed within the print_data function.
3. Print an error message if an unsupported type specifier is passed (e.g., 'x').

```
*/
#include <stdio.h>

void print_data(void* data, char type);

int main() {
    int num = 10;
    float pi = 3.14;
    char *message = "abhi";

    print_data(&num, 'i');
    print_data(&pi, 'f');
    print_data(message, 's');

    return 0;
}

void print_data(void* data, char type) {
    switch (type) {
        case 'i':
            printf("Integer: %d\n", *(int*)data);
            break;
        case 'f':
            printf("Float: %.2f\n", *(float*)data);
            break;
        case 's':
            printf("String: %s\n", (char*)data);
            break;
        default:
            printf("Error: Unsupported type specifier '%c'\n", type);
            break;
    }
}
```



Requirements

- In this challenge, you are going to write a program that tests your understanding of char arrays

- write a function to count the number of characters in a string (length)
  - cannot use the strlen library function
  - function should take a character array as a parameter
  - should return an int (the length)

- write a function to concatenate two character strings
  - cannot use the strcat library function
  - function should take 3 parameters
    - char result[]
    - const char str1[]
    - const char str2[]
    - can return void

- write a function that determines if two strings are equal
  - cannot use strcmp library function
  - function should take two const char arrays as parameters and return a Boolean of true if they are equal and false otherwise

```
#include <stdio.h>
void my_strcat(char *, char *, char *);
int my_strcmp(char *, char *);
int my_strlen(char *);
int main()
{
  //char option;

  char str1[50], str2[50], result[50];

  printf("Enter the 1st string: ");
  scanf("%[^\n]", str1);
  getchar();
  printf("Enter the 2nd string: ");
```

```
  scanf("%[^\n]", str2);

  my_strcat(str1, str2, result);
  printf("The concatenated string is: %s\n", result);

  int res = my_strcmp(str1, str2);
  res == 0 ? printf("Strings are equal\n") : printf("Strings are not equal\n");

  printf("The length of 1st string is %d\n", my_strlen(str1));
  printf("The length of 2nd string is %d\n", my_strlen(str2));
  printf("The length of concatenated string is %d\n", my_strlen(result));

}

void my_strcat(char *str1, char *str2, char *result)
{
   int i=0;
   while(str1[i])
   {
       result[i]=str1[i];
       ++i;
   }
   int j=0;
   while(str2[j])
   {
       result[i+j]=str2[j];
       ++j;
   }
}

int my_strcmp(char str1[], char str2[])
{
    while(*str1 && *str2)
    {
        if(*str1 != *str2)
        return 1;

        str1++;
        str2++;
    }
    return 0;
}

int my_strlen(char str[])
```

```
{
    int i=0;
    while(str[++i]);
    return i;
}
```