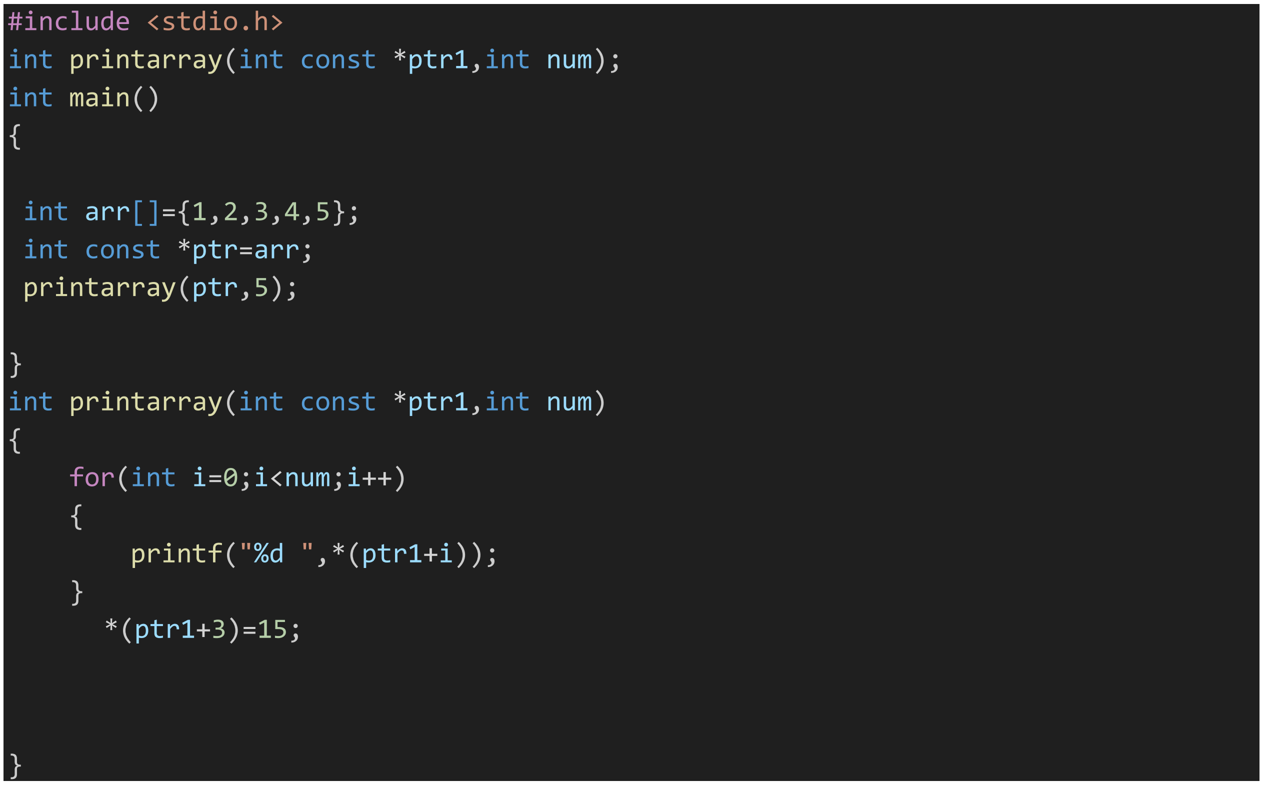
Problem 1: Array Element Access
Write a program in C that demonstrates the use of a pointer to a const array of integers. The program should do the following:
1. Define an integer array with fixed values (e.g., {1, 2, 3, 4, 5}).
2. Create a pointer to this array that uses the const qualifier to ensure 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.



```c
#include <stdio.h>
int printarray(int const *ptr1,int num);
int main()
{

 int arr[]={1,2,3,4,5};
 int const *ptr=arr;
 printarray(ptr,5);

}
int printarray(int const *ptr1,int num)
{
    for(int i=0;i<num;i++)
    {
        printf("%d ",*(ptr1+i));
    }
      *(ptr1+3)=15;



}
```

Output

```
1 2 3 4 5
```

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 num=10,num1=20;
 int const *ptr=&num;
 int *const ptr1=&num1;
 printf("address of ptr= %p \n",&ptr);
 printf("value at ptr =%d \n",*ptr);
 printf("address of ptr1= %p \n",&ptr1);
 printf("value at ptr1 =%d \n",*ptr1);
 *ptr=30;
 ptr1=&num;
return 0;
}
```

Write a program to find the length of the string

```
#include<stdio.h>
int main()
{
    char str1[]="hi there";
    char str2[]="welcome";
    int count=0;
    while(str1[count]!='\0')
        count++;
    printf("length of str1 is %d \n",count);
    count=0;
    while(str2[count]!='\0')
        count++;
    printf("length of str12 is %d \n",count);
}
```

Output

```
length of str1 is 8
length of str12 is 7
```

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:

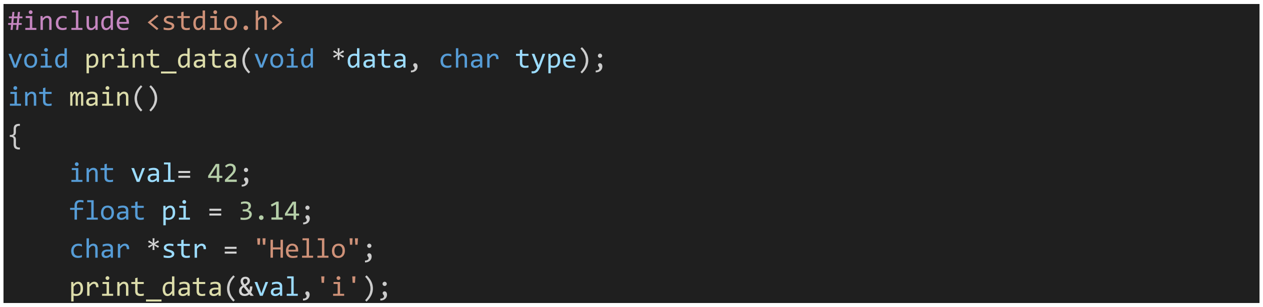
Integer: 42

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').



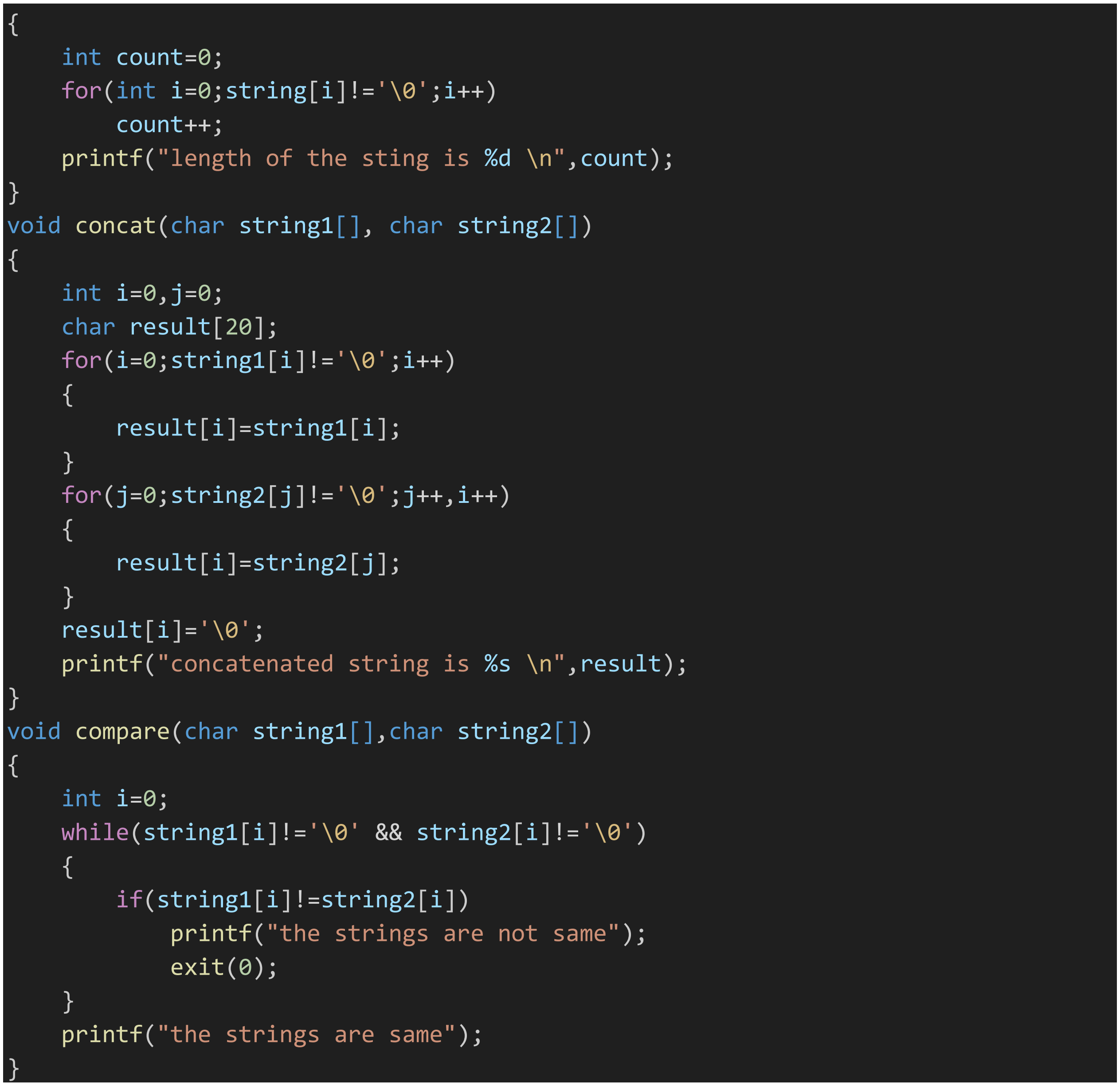
```c
#include <stdio.h>
void print_data(void *data, char type);
int main()
{
    int val= 42;
    float pi = 3.14;
    char *str = "Hello";
    print_data(&val,'i');
    print_data(&pi,'f');
    print_data(str,'s');
    return 0;
}
void print_data(void *data, char type)
{
    switch (type)
    {
        case 'i':
            printf("integer: %d \n",*(int*)data);
            break;
        case 'f':
            printf("float: %0.2f \n",*(float*)data);
            break;
        case 's':
            printf("string: %s \n",*(char*)data);
            break;
        default:
            printf("Error\n", type);
    }
}
```

Output

```
Integer: 42
Float: 3.14
String: Hello
```

```
#include<stdio.h>
#include<stdlib.h>
void length(char string[]);
void concat(char string1[],char string2[]);
void compare(char string1[],char string2[]);
int main()
{
    char str1[]="vivek";
    char str2[]="vivek1";
    length(str1);
    length (str2);
    concat(str1,str2);
    compare(str1,str2);

}
void length(char string[])
```



```
{
    int count=0;
    for(int i=0;string[i]!='\0';i++)
        count++;
    printf("length of the sting is %d \n",count);
}
void concat(char string1[], char string2[])
{
    int i=0,j=0;
    char result[20];
    for(i=0;string1[i]!='\0';i++)
    {
        result[i]=string1[i];
    }
    for(j=0;string2[j]!='\0';j++,i++)
    {
        result[i]=string2[j];
    }
    result[i]='\0';
    printf("concatenated string is %s \n",result);
}
void compare(char string1[],char string2[])
{
    int i=0;
    while(string1[i]!='\0' && string2[i]!='\0')
    {
        if(string1[i]!=string2[i])
            printf("the strings are not same");
            exit(0);
    }
    printf("the strings are same");
}
```