```
/*
int const* ===>value becomes constant but the pointer is modifiable
int *const  ===>value become modifiable but the pointer becomes constant
int const * const ===> both are unalterable
*/
#include <stdio.h>

int main()
{
   int num = 800;
   printf("001num = %d \n",num);
   int const *const pNum = &num;
   printf("001pNum = %p \n",pNum);

   int num1 = 900;
   pNum = &num1;

   return 0;
}
```

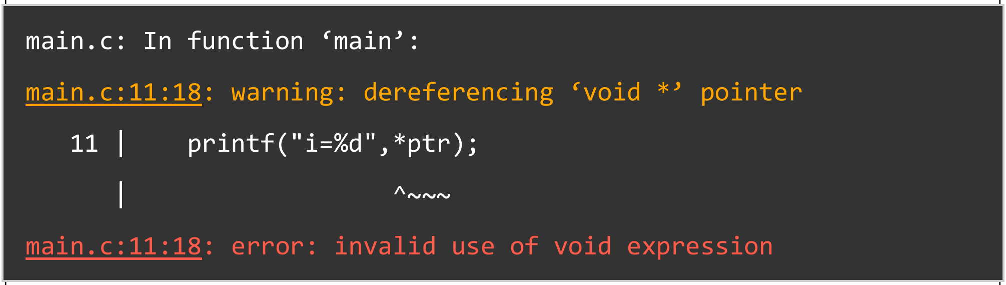
10:04has context menu

VOID POINTER

```
#include <stdio.h>

int main()
{
  int i=1234;
  float pi=3.14;
  char c='A';
  void *ptr;
  ptr=&i;
  printf("i=%d",*ptr);

   return 0;
 }
```



```
main.c: In function ‘main’:
main.c:11:18: warning: dereferencing ‘void *’ pointer
   11 |     printf("i=%d",*ptr);
      |                   ^~~~
main.c:11:18: error: invalid use of void expression
```

error bcoz compiler is confused of dat type

```
#include <stdio.h>

int main()
{
  int i=1234;
  float pi=3.14;
  char c='A';
  void *ptr;
  ptr=&i;
  printf("i=%d",*(int *)ptr);

   return 0;
}
```

```
i=1234

...Program finished with exit code 0
Press ENTER to exit console.
```

no error bcoz void pointer was **type casted** to interger;

```
#include <stdio.h>

int main()
{
  int i=1234;
  float pi=3.14;
  char c='A';
  void *ptr;
  ptr=&i;
  printf("i=%d\n",*(int *)ptr);
  ptr=&pi;
  printf("pi=%f\n",*(float *)ptr);
  ptr=&c;
  printf("c=%c\n",*(char *)ptr);
   return 0;
}
```

```
i=1234
pi=3.140000
c=A


...Program finished with exit code 0
Press ENTER to exit console.
```

Arrays-pointers

```
#include <stdio.h>

int main()
{
  int a[]={1,2,3};
  int *ptr=a;// or like this- int*ptr=&a[0]
  printf("address of a[0]=%p\n",a);
  printf("ptr=%p\n",ptr);
}
```

```
address of a[0]=0x7ffc2b1b80bc
ptr=0x7ffc2b1b80bc


...Program finished with exit code 0
Press ENTER to exit console.
```

Array in function

```
#include <stdio.h>
int addArray(int array[],int);
int main()
{
  int a[]={1,2,3,4,5,6,7,8,9,10};
  int sum=0;
  sum=addArray(a,10);
  printf("sum=%d\n",sum);
  return 0;;
}
int addArray(int y[],int n){
   int arsum=0;
   for(int i=0;i<n;i++){
      arsum=arsum+y[i];
   }

return arsum;
}
```

Array in fuction using pointers

```
#include <stdio.h>
int addArray(int *,int);
int main()
{
  int a[]={1,2,3,4,5,6,7,8,9,10};
  int sum=0;
  sum=addArray(a,10);//or addArray(&a[0],10)
  printf("sum=%d\n",sum);
  return 0;;
}
int addArray(int *y,int n){
   int arsum=0;
   for(int i=0;i<n;i++){
      arsum=arsum+*(y+i);
   }

return arsum;
}
```

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 printArray(const int *,int);
int main()
{
  int a[]={1,2,3,4,5};

  printArray(a,5);

  return 0;
}
```

```
void printArray(const int *y,int n){
  printf("elements of array is :\n");
  for(int i=0;i<n;i++){
    printf("a[%d]=%d ",i,*(i+y));
  }
  y[3]=6;
}-------------------------------------------------------------------------------
```

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;
  int a;
  int const *const ptr=&num;
  printf("num=%d\n",num);
  printf("address of num=%p\n",&num);
  printf("value in ptr=%p\n",ptr);
  printf("value pointed by ptr=%d\n",*ptr);
  //*ptr=20;
  ptr=&a;

}
```

String

```
#include <stdio.h>

int main()
{
  char str1[]="antony";
  char str2[]="petta";
  int count=0;
  while(str1[count]!='\0'){
    count++;
  }
  printf("length of string1 is %d\n",count);
  count=0;
  while(str2[count]!='\0'){
    count++;
  }
  printf("length of string2 is %d",count);

}
```

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').
15:26has context menu
****

```
#include <stdio.h>
void print_data(void*,char);
int main()
{
    int a=10;
    print_data(&a,'i');
    float f=24.567;
    print_data(&f,'f');
    char s[]="antony";
    print_data(s,'s');
    print_data(a,'a');
}
void print_data(void *data,char type){
    switch (type){
        case 'i':
        printf("this is an integer value:");
        printf("%d\n",*(int*)data);
        break;
        case 'f':
        printf("this is an float value:");
        printf("%f\n",*(float*)data);
        break;
        case 's':
        printf("this is an string value:");
        printf("%s\n",(char*)data);
        break;
        default:
        printf("unsupported data type passed");

    }
}
```

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 strien 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 str21]

can retum void

write a function that determines if two strings are equal

cannot use stromp library function

function should take two const char arrays as parameters and retum a Boolean of true if they are equal and false otherwise
****

```
#include <stdio.h>
int strlength(char *);
void strconcat(char *,char *);
int strcomp(char*,char*);
void main(){
  char str1[20],str2[20];
  printf("enter string 1 :");
  scanf("%s",str1);
  printf("enter string 2 :");
  scanf("%s",str2);
  printf("length of string 1:%d",strlength(str1));

  printf("\n");

  printf("length of string 2:%d",strlength(str2));

  printf("\n");
    int value=strcomp(str1,str2);
  if(1==value){
    printf("strings are same");
  }
  else{
    printf("strins are not same");
  }
   printf("\n");
  //concatenation
  printf("after concatenation :");
  strconcat(str1,str2);



}
int strlength(char *str1){
  int length=0;
  while(str1[length]!='\0'){
    length++;
  }
  return length;

}
void strconcat(char *str1,char *str2){
  int length2=strlength(str2);
  int length1=strlength(str1);
  for(int i=0;i<length2;i++){
    str1[i+length1]=str2[i];
  }
  str1[length1+length2]='\0';
  printf("\n%s",str1);
}
int strcomp(char *str1,char *str2){
  int i=0;
  while(str1[i] != '\0' && str2[i] != '\0'){
    if(str2[i]!=str1[i]){
        return 0;
        }
        i++;
    }
    if(str1[i] == '\0' && str2[i] == '\0'){
        return 1;
    }
    return 0;
}
```
