int pointer cannot point to a floatvariable.

 For example, suppose you are required to design a function that can accept any type value as argument, process data and return results. For such situation, you need a pointer that must work with all types. A void pointer is a special pointer that can point to object of any type

**Dereferencing** is the process of retrieving data from memory location pointed by a pointer. It converts block of raw memory bytes to a meaningful data (data is meaningful if **type** is associated).

To dereference a void pointer you must [typecast](https://codeforwin.org/2017/08/typecasting-c-programming.html) it to a valid pointer type.

int num = 10;

void \* vPtr = &num; // void pointer pointing at num

int value = \*((int \*) vPtr); // Dereferencing void pointer

To perform pointer arithmetic on void pointer you must first typecast to other type.

int arr[] = {10, 20, 30, 40, 50};

void \* vPtr = &arr; // void pointer pointing at arr

vPtr = ((int \*) vPtr + 1); // add 1 to void pointer

See following program : we can pass address of first element in the array to function and use of switch case.

Write a C function to [accept an array and print its elements](https://codeforwin.org/2015/07/c-program-to-read-and-print-elements-in-array.html). The function must accept array of different types.

#include <stdio.h>

#define SIZE 10

void printArray(void \* vPtr, int size, int type);

int main()

{

int num[SIZE] = {10, 20,30};

float fractional[SIZE] = {1.1f, 1.2f, 1.3f};

char characters[SIZE] = {'C', 'o', 'd', 'e'};

printf("\nElements of integer array: ");

printArray(&num, SIZE, 1);

printf("\nElements of float array: ");

printArray(&fractional, SIZE, 2);

printf("\nElements of character array: ");

printArray(&characters, SIZE, 3);

return 0;

}

void printArray(void \* vPtr, int size, int type)

{

int i;

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

{

switch(type)

{

case 1:

printf("%d, ", \*((int \*)vPtr + i));

break;

case 2:

printf("%f, ", \*((float \*)vPtr + i));

break;

case 3:

printf("%c, ", \*((char \*)vPtr + i));

break;

}

}

}

**Output**

Elements of integer array: 10, 20, 30, 40, 50, 60, 70, 80, 90, 100,

Elements of float array: 1.100000, 1.200000, 1.300000, 1.400000, 1.500000, 1.600000, 1.700000, 1.800000, 1.900000, 2.000000,

Elements of character array: C, o, d, e, f, o, r, w, i, n,

[**Constant pointer**](https://codeforwin.org/2017/11/constant-pointer-and-pointer-to-constant-in-c.html#const-pointer)

[**Pointer to constant**](https://codeforwin.org/2017/11/constant-pointer-and-pointer-to-constant-in-c.html#pointer-to-const)

[**Constant pointer to constant**](https://codeforwin.org/2017/11/constant-pointer-and-pointer-to-constant-in-c.html#const-pointer-to-const)

**1.Constant Pointer**

Constant Pointer is a pointer variable whose value cannot be altered throughout the program.

**Note:** You must initialize a constant pointer at the time of its declaration*.*

#include <stdio.h>

int main()

{

int num1, num2;

int \* const const\_ptr = &num1; //// Constant pointer to num1

\*const\_ptr = 10; // Modification of value pointed by pointer is allowed

const\_ptr = &num2; // Error, // Modification of pointer value is not allowed

printf("Num1 = %d\n", num1);

printf("Num1 = %d\n", \*const\_ptr);

return 0;

}

**2.Pointer To Constant**

You can modify pointer value, but you cannot modify the value pointed by pointer.

#include <stdio.h>

int main()

{

int num = 10;

const int \* ptr\_const; // Pointer to constant

ptr\_const = &num; // ptr\_const points to num

num = 20; // Direct modification on num is allowed

\*ptr\_const = 100; // Error, Modification of value pointed by pointer is not allowed

printf("Num = %d\n", num);

printf("Num = %d\n", \*ptr\_const);

return 0;

}

Pointer to constant does not allows you to modify the pointed value, using pointer. However, you can directly perform modification on variable (without using pointer).

For example,

int num = 10;

const int \* ptr = &num;

// Modification of pointed value

// using pointer is not allowed

\*ptr = 20; // Error

// You can still modify the pointed value directly

num = 20; // Works

**3.Constant Pointer To Constant**

A constant pointer to constant is a combination of [constant pointer](https://codeforwin.org/2017/11/constant-pointer-and-pointer-to-constant-in-c.html#const-pointer) and [pointer to constant](https://codeforwin.org/2017/11/constant-pointer-and-pointer-to-constant-in-c.html#pointer-to-const). It is a pointer that does not allow modification of pointer value as well as value pointed by the pointer.

#include <stdio.h>

int main()

{

int num1 = 10;

int num2 = 20;

const int \* const ptr = &num1; // Declare constant pointer pointing at num1

ptr = &num2; // Error

\*ptr = 100; // Error

printf("Num1 = %d\n", num1);

printf("Num1 = %d\n", \*ptr);

return 0;

}

| **Constant pointer** | **Pointer to constant** | **Constant pointer to constant** |
| --- | --- | --- |
| Pointer value cannot be modified | Pointer value can be modified | Pointer value cannot be modified |
| Value pointed by the pointer can be modified | Value pointed by the pointer cannot be modified | Value pointed by the pointer cannot be modified |