viettel

Theo cách của bạn

THE C PROGRAMMING

Presenter: Trinh Cao Cuong

3/2024

TABLE OF CONTENTS

1. MEMORY LAYOUT*
2. VARIABLE AND OPERATOR*
3. POINTER & ARRAY
4. FUNCTION, INLINE FUNCTION, MARCO
5. STRUCT, ENUM, UNION
6. SUMMARY

* Advanced only

1. MEMORY LAYOUT

viettel

Theo cách của bạn

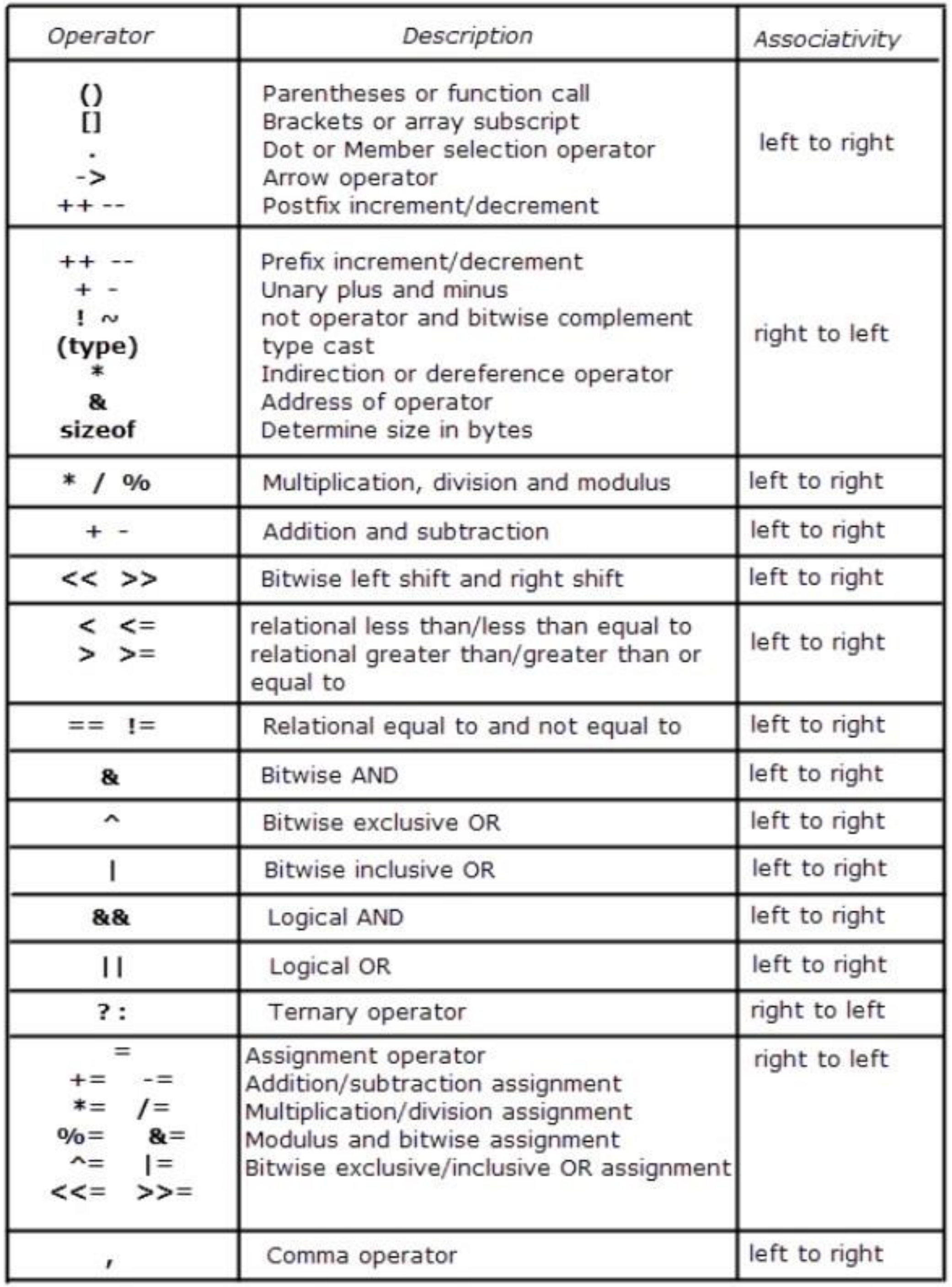
2. VARIABLE & OPERATOR

Theo cách của bạn

VARIABLE & OPERATOR : OPERATOR IN C

Table beside describes the precedence order and associativity of operators in C.
The precedence of the operator **decreases from top to bottom.**

Operator Precedence and Associativity is the concept that decides which operator will be evaluated first in the case when there are multiple operators present in an expression.



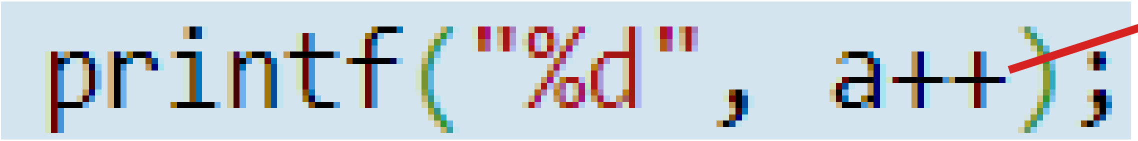
Operator	Description	Associativity
() [] . -> ++ --	Parentheses or function call Brackets or array subscript Dot or Member selection operator Arrow operator Postfix increment/decrement	left to right
++ -- + - ! ~ (type) * & sizeof	Prefix increment/decrement Unary plus and minus not operator and bitwise complement type cast Indirection or dereference operator Address of operator Determine size in bytes	right to left
* / %	Multiplication, division and modulus	left to right
+ -	Addition and subtraction	left to right
<< >>	Bitwise left shift and right shift	left to right
< <= > >=	relational less than/less than equal to relational greater than/greater than or equal to	left to right
== !=	Relational equal to and not equal to	left to right
&	Bitwise AND	left to right
^	Bitwise exclusive OR	left to right
|	Bitwise inclusive OR	left to right
&&	Logical AND	left to right
||	Logical OR	left to right
? :	Ternary operator	right to left
= += -= *= /= %= &= ^= |= <<= >>=	Assignment operator Addition/subtraction assignment Multiplication/division assignment Modulus and bitwise assignment Bitwise exclusive/inclusive OR assignment	right to left
,	Comma operator	left to right

VARIABLE & OPERATOR : POSTFIX ++ --

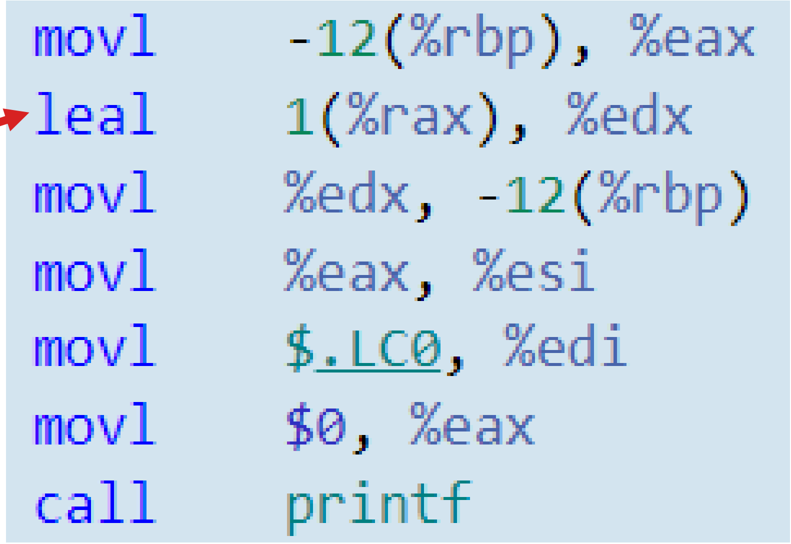
The postfix ***a++, a--*** work as below:

- Create temp value of **a**
- Increase value of **a**
- Return temp value to printf

a-- is the same work flow



```
printf("%d", a++);
```



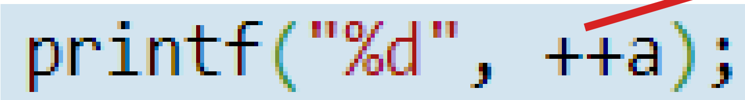
```
movl     -12(%rbp), %eax
leal     1(%rax), %edx
movl     %edx, -12(%rbp)
movl     %eax, %esi
movl     $.LC0, %edi
movl     $0, %eax
call     printf
```

VARIABLE & OPERATOR : PREFIX ++ --

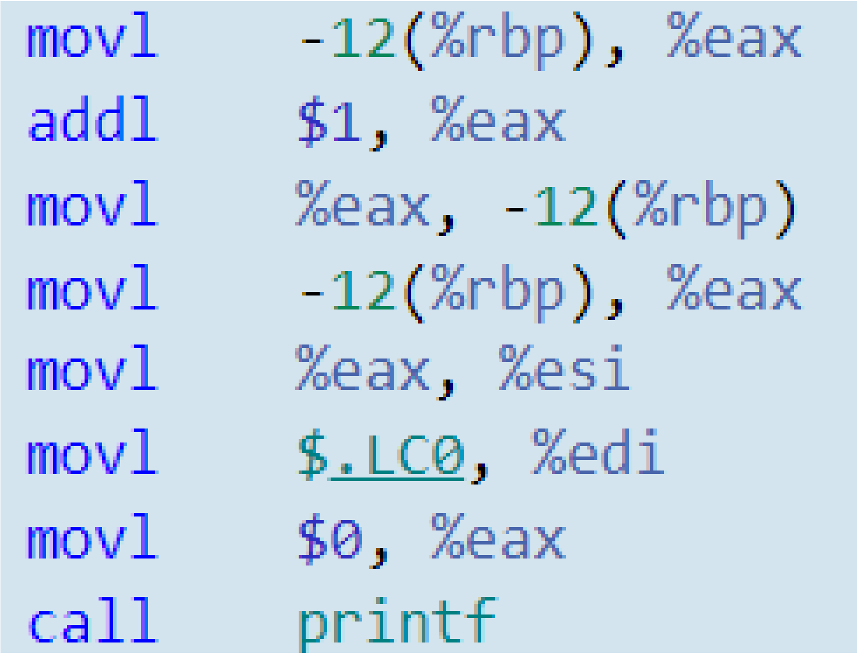
The postfix ***++a, --a*** work as below:

- Increase value of **a**
- Return **a** value to printf

--a is the same work flow



```
printf("%d", ++a);
```

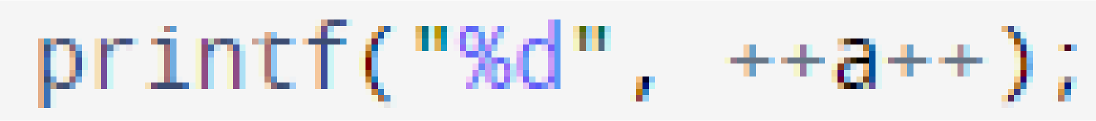


```
movl     -12(%rbp), %eax
addl     $1, %eax
movl     %eax, -12(%rbp)
movl     -12(%rbp), %eax
movl     %eax, %esi
movl     $.LC0, %edi
movl     $0, %eax
call     printf
```

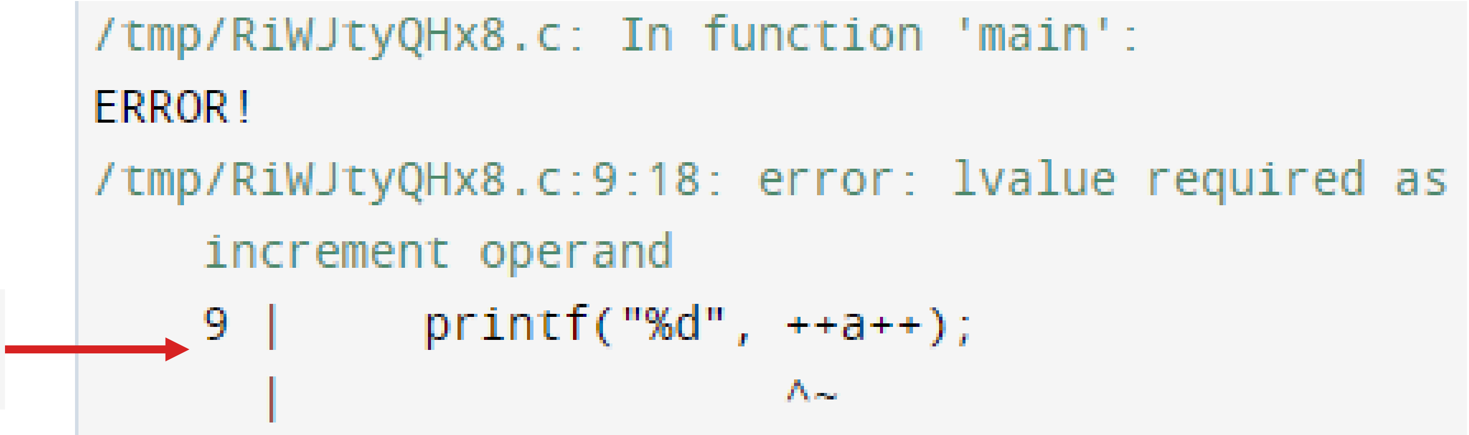
VARIABLE & OPERATOR : PREFIX COMBINE WITH POSTFIX

Prefix and postfix return value in register but not address (***lvalue error***)

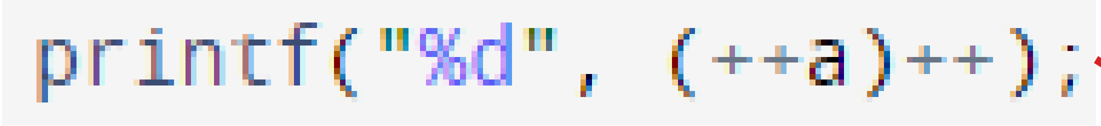
Postfix has more precedence than prefix



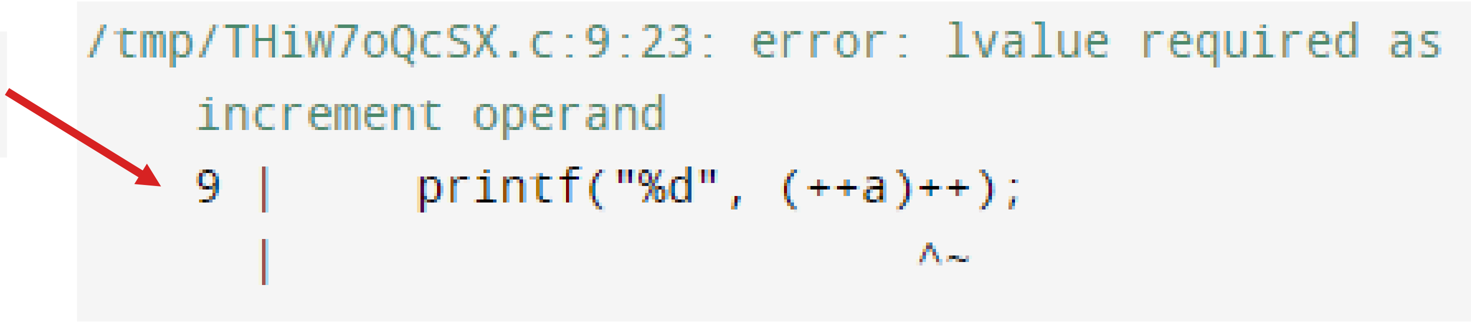
```
printf("%d", ++a++);
```



```
/tmp/RiWJtyQHx8.c: In function 'main':
ERROR!
/tmp/RiWJtyQHx8.c:9:18: error: lvalue required as
    increment operand
    9 |     printf("%d", ++a++);
      |                    ^~
```



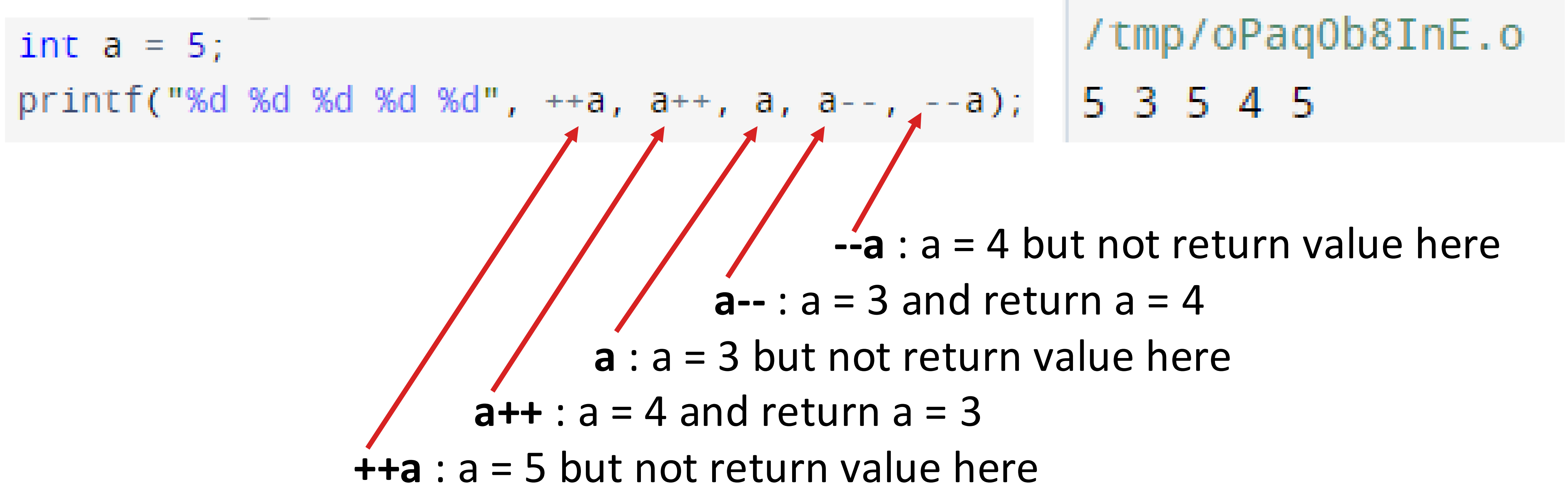
```
printf("%d", (++a)++);
```



```
/tmp/THiw7oQcSX.c:9:23: error: lvalue required as
    increment operand
    9 |     printf("%d", (++a)++);
      |                       ^~
```

VARIABLE & OPERATOR : PREFIX COMBINE WITH POSTFIX

() : Ordering from right to left



```c
int a = 5;
printf("%d %d %d %d %d", ++a, a++, a, a--, --a);
```

```
/tmp/oPaq0b8InE.o
5 3 5 4 5
```

--a : a = 4 but not return value here

a-- : a = 3 and return a = 4

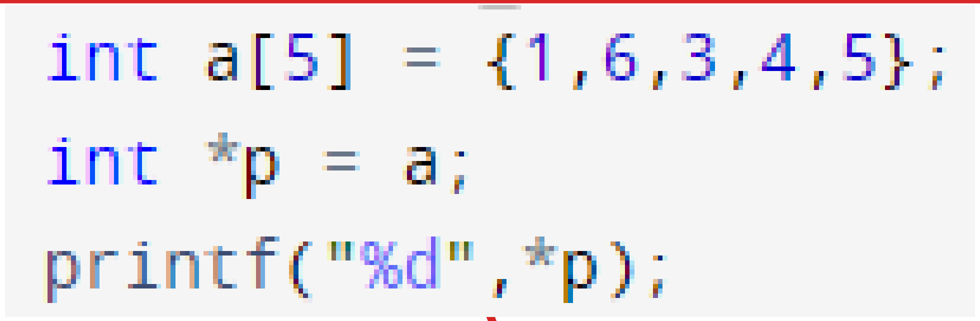
a : a = 3 but not return value here

a++ : a = 4 and return a = 3

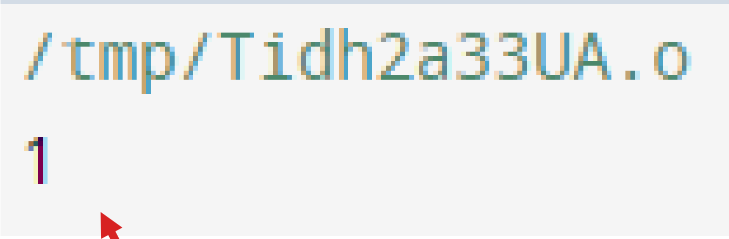
++a : a = 5 but not return value here

Next: return value to **--a = a = ++a = 5**

VARIABLE & OPERATOR : PREFIX ,POSTFIX COMBINE WITH *



```c
int a[5] = {1,6,3,4,5};
int *p = a;
printf("%d",*p);
```



```
/tmp/Tidh2a33UA.o
1
```

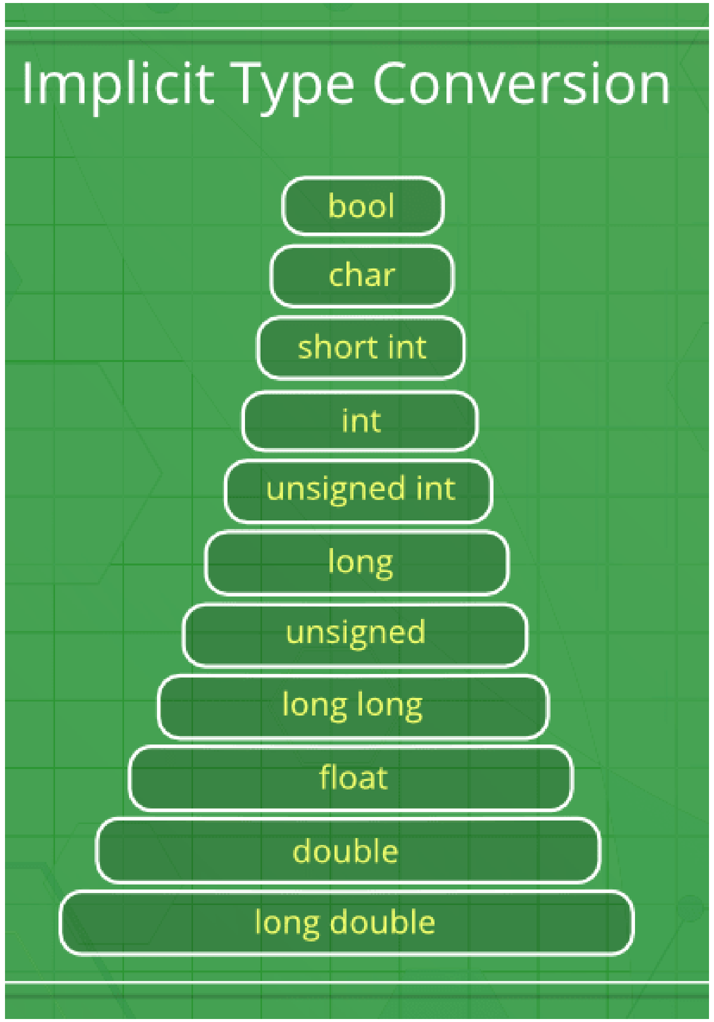
*p	1
++*p	2
++(*p)	2
*++p	6
*(++p)	6
*p++	1
*(p++)	1

VARIABLE & OPERATOR : TYPE CASTING

Implicit type conversion: Done by compiler on its own, All the data types of the variables are upgraded to the data type of the variable with the largest data type

Explicit type conversion: (type) expression

It return a temp value



Implicit Type Conversion

bool

char

short int

int

unsigned int

long

unsigned

long long

float

double

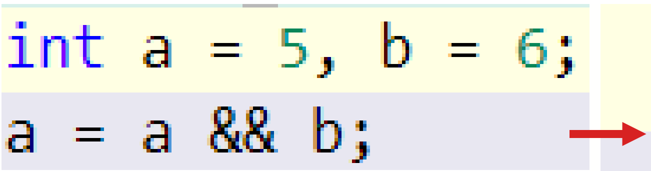
long double

VARIABLE & OPERATOR : LOGIC OPERATOR

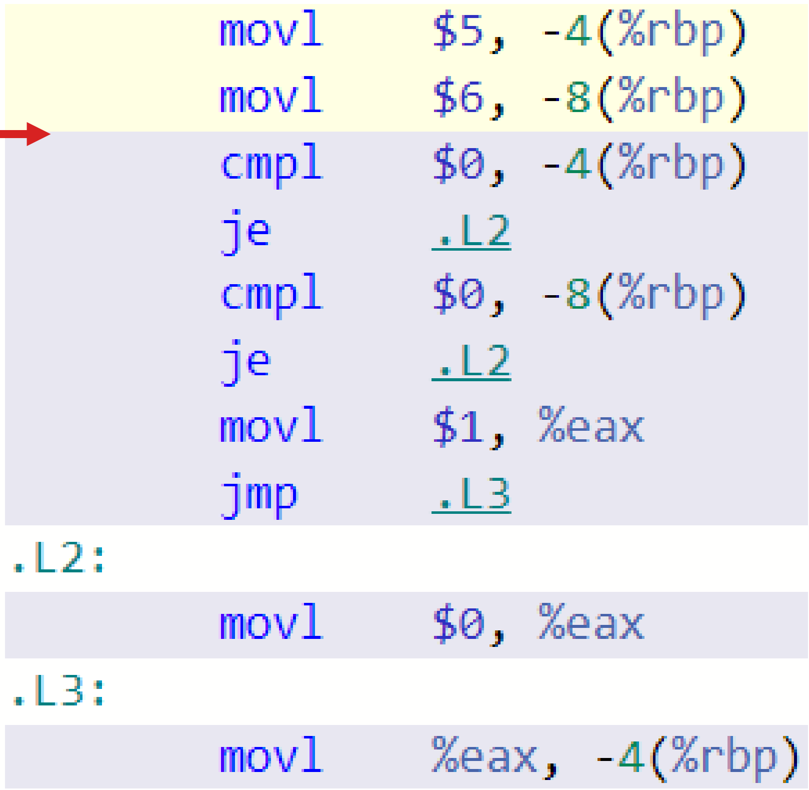
Logic operator will check:

- ||: if argument = 1 then jump
- &&: if argument = 0 then jump

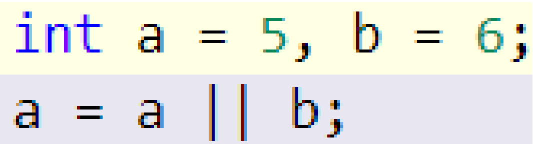
This feature in order to reduce stress and error



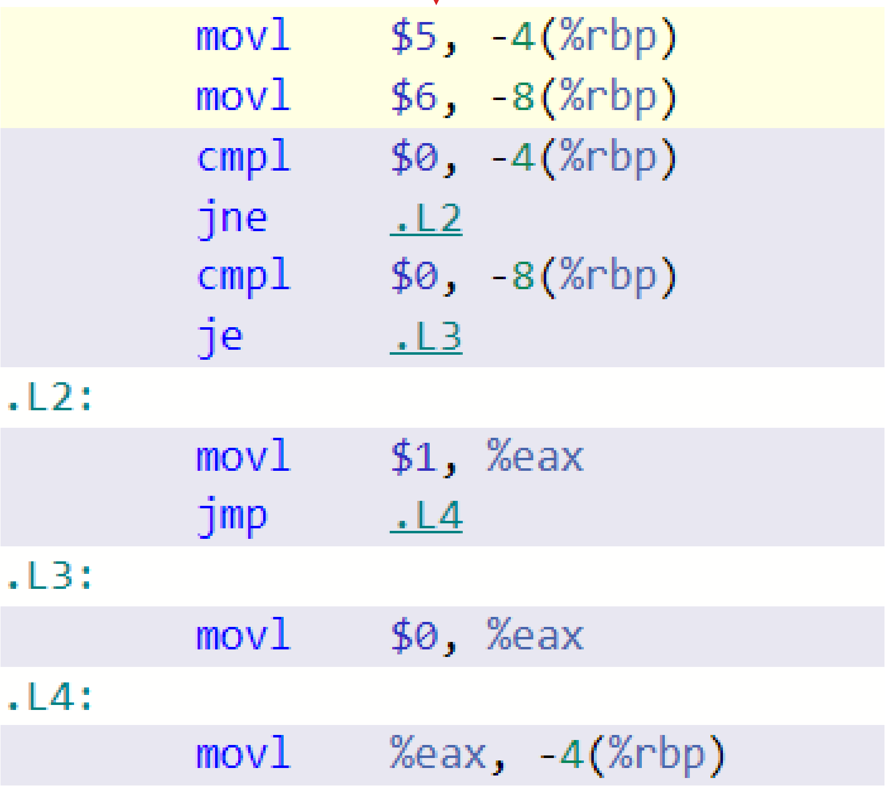
```
int a = 5, b = 6;
a = a && b;
```



```
        movl    $5, -4(%rbp)
        movl    $6, -8(%rbp)
        cmpl    $0, -4(%rbp)
        je      .L2
        cmpl    $0, -8(%rbp)
        je      .L2
        movl    $1, %eax
        jmp     .L3
.L2:
        movl    $0, %eax
.L3:
        movl    %eax, -4(%rbp)
```



```
int a = 5, b = 6;
a = a || b;
```



```
        movl    $5, -4(%rbp)
        movl    $6, -8(%rbp)
        cmpl    $0, -4(%rbp)
        jne     .L2
        cmpl    $0, -8(%rbp)
        je      .L3
.L2:
        movl    $1, %eax
        jmp     .L4
.L3:
        movl    $0, %eax
.L4:
        movl    %eax, -4(%rbp)
```

3. POINTER & ARRAY

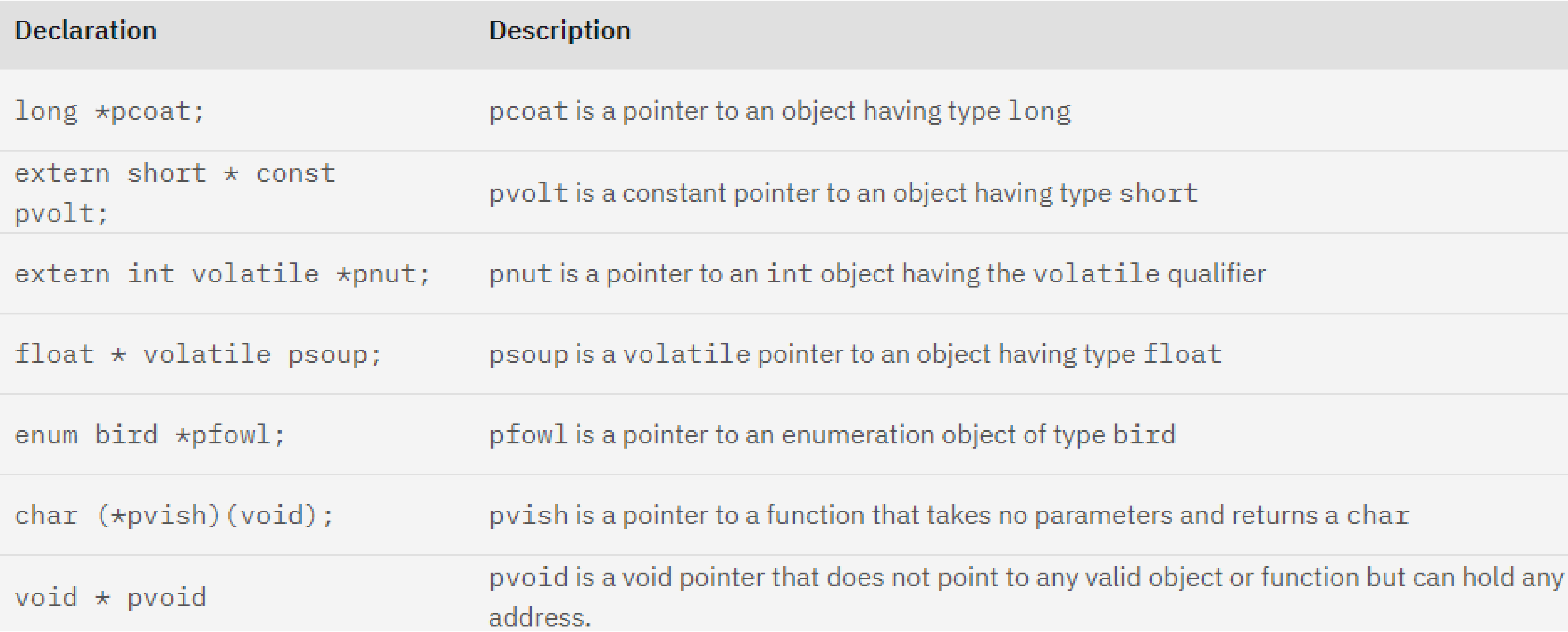
POINTER & ARRAY : POINTER DECLARE

A **pointer** type variable holds the address of a data object or a function.

Note that the placement of the type qualifiers **volatile** and **const** affects the semantics of a pointer declaration.
If either of the qualifiers appears before the *, the declarator describes a **pointer to a type-qualified object**. If either of the qualifiers appears between the * and the identifier, the declarator describes a **type-qualifed pointer**.

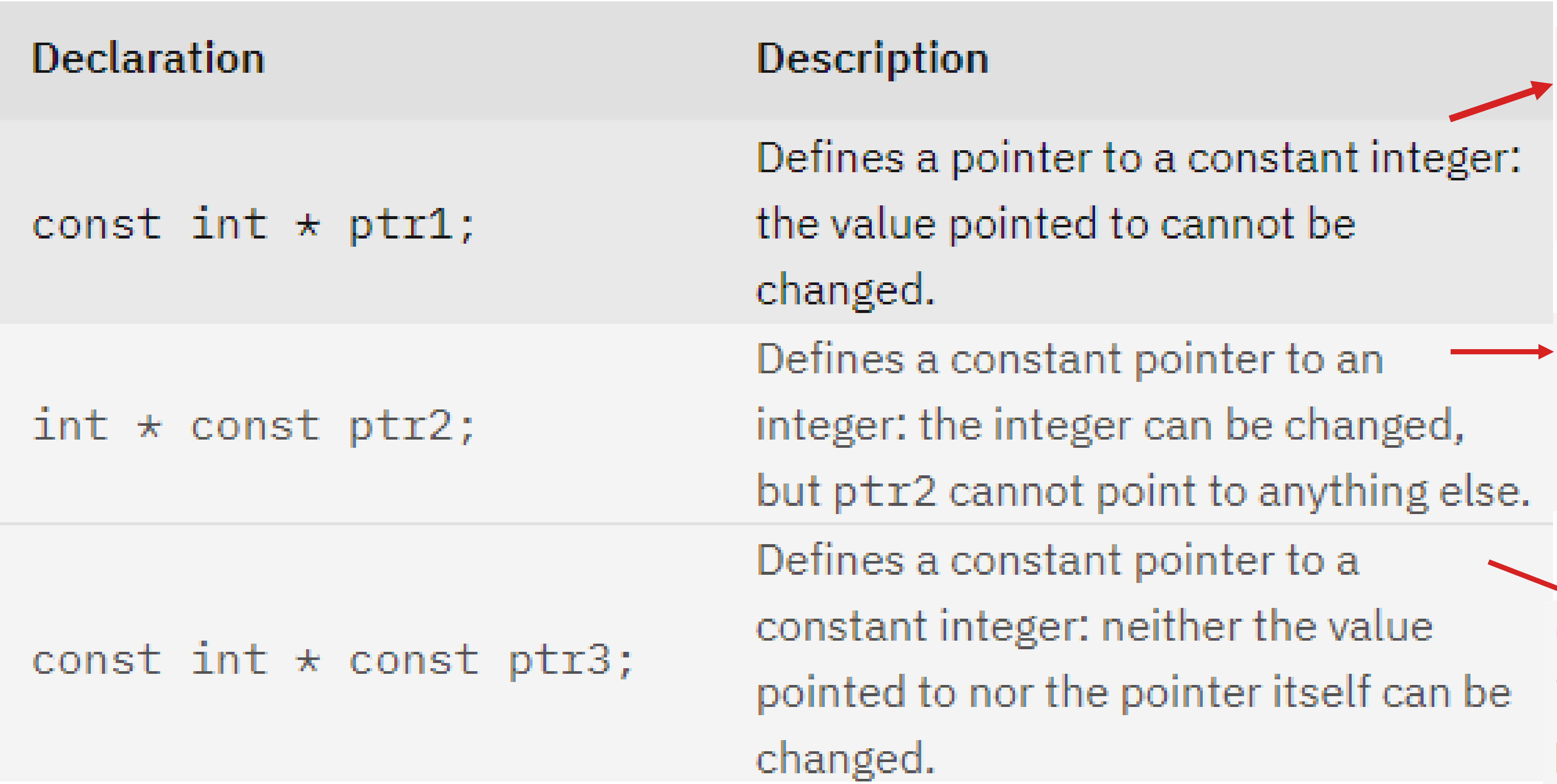
Pointer is a address value and store in memory by 64-bits or 32-bits (depend on OS), same as type long long int

POINTER & ARRAY : POINTER DECLARE

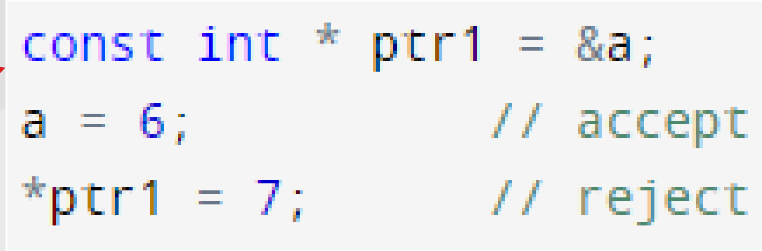


Declaration	Description
`long *pcoat;`	pcoat is a pointer to an object having type `long`
`extern short * const pvolt;`	pvolt is a constant pointer to an object having type `short`
`extern int volatile *pnut;`	pnut is a pointer to an `int` object having the `volatile` qualifier
`float * volatile psoup;`	psoup is a `volatile` pointer to an object having type `float`
`enum bird *pfowl;`	`pfowl` is a pointer to an enumeration object of type `bird`
`char (*pvish)(void);`	pvish is a pointer to a function that takes no parameters and returns a `char`
`void * pvoid`	pvoid is a void pointer that does not point to any valid object or function but can hold any address.

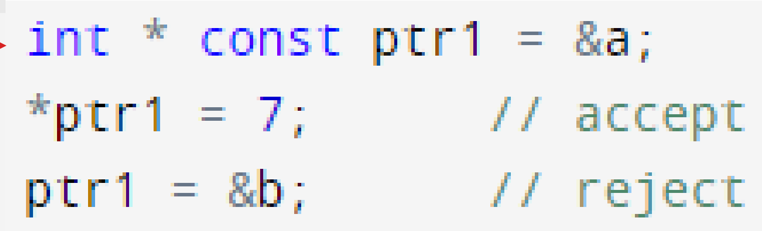
POINTER & ARRAY : CONST POINTER DECLARE



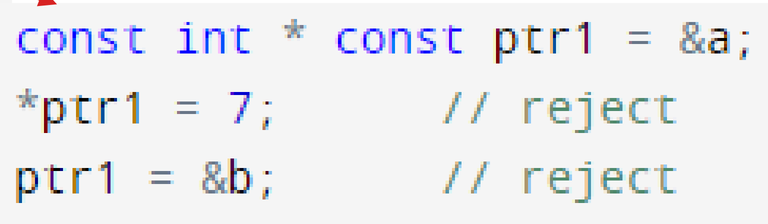
Declaration	Description
`const int * ptr1;`	Defines a pointer to a constant integer: the value pointed to cannot be changed.
`int * const ptr2;`	Defines a constant pointer to an integer: the integer can be changed, but ptr2 cannot point to anything else.
`const int * const ptr3;`	Defines a constant pointer to a constant integer: neither the value pointed to nor the pointer itself can be changed.



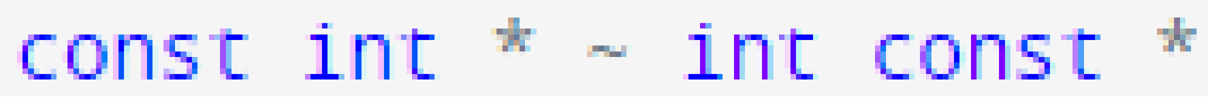
```
const int * ptr1 = &a;
a = 6;              // accept
*ptr1 = 7;          // reject
```



```
int * const ptr1 = &a;
*ptr1 = 7;          // accept
ptr1 = &b;          // reject
```



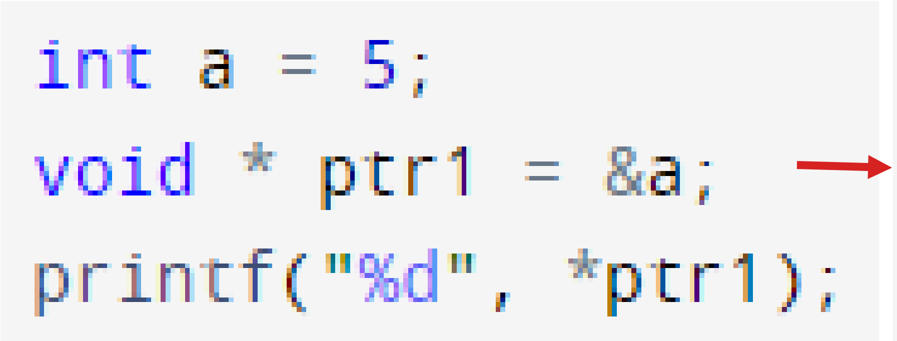
```
const int * const ptr1 = &a;
*ptr1 = 7;          // reject
ptr1 = &b;          // reject
```



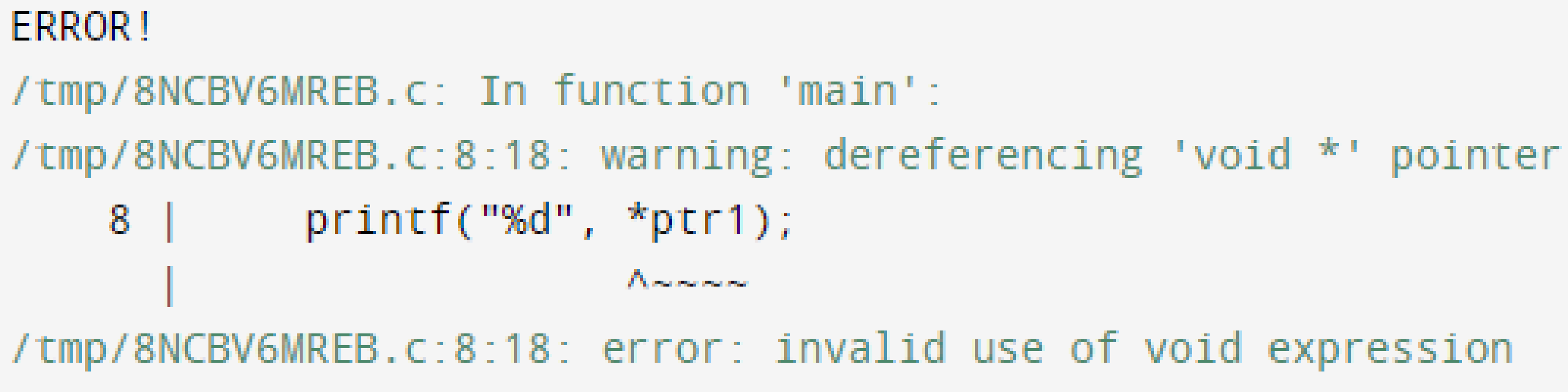
```
const int * ~ int const *
```

POINTER & ARRAY : VOID POINTER DECLARE

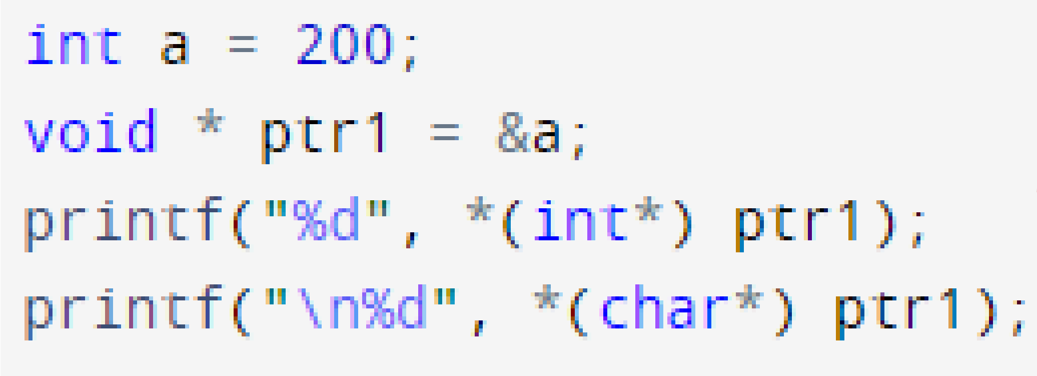
Void pointer hold a address but couldn’t read or write value pointed.
User can casting type of **void pointer** to using.



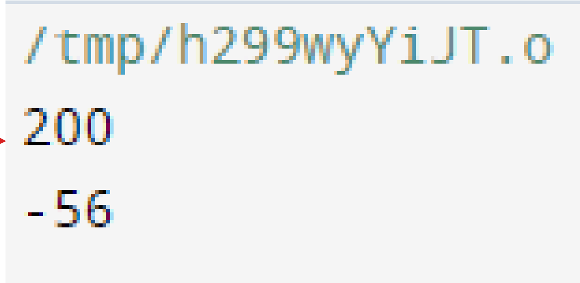
```
int a = 5;
void * ptr1 = &a;
printf("%d", *ptr1);
```



```
ERROR!
/tmp/8NCBV6MREB.c: In function 'main':
/tmp/8NCBV6MREB.c:8:18: warning: dereferencing 'void *' pointer
    8 |     printf("%d", *ptr1);
      |                  ^~~~~
/tmp/8NCBV6MREB.c:8:18: error: invalid use of void expression
```

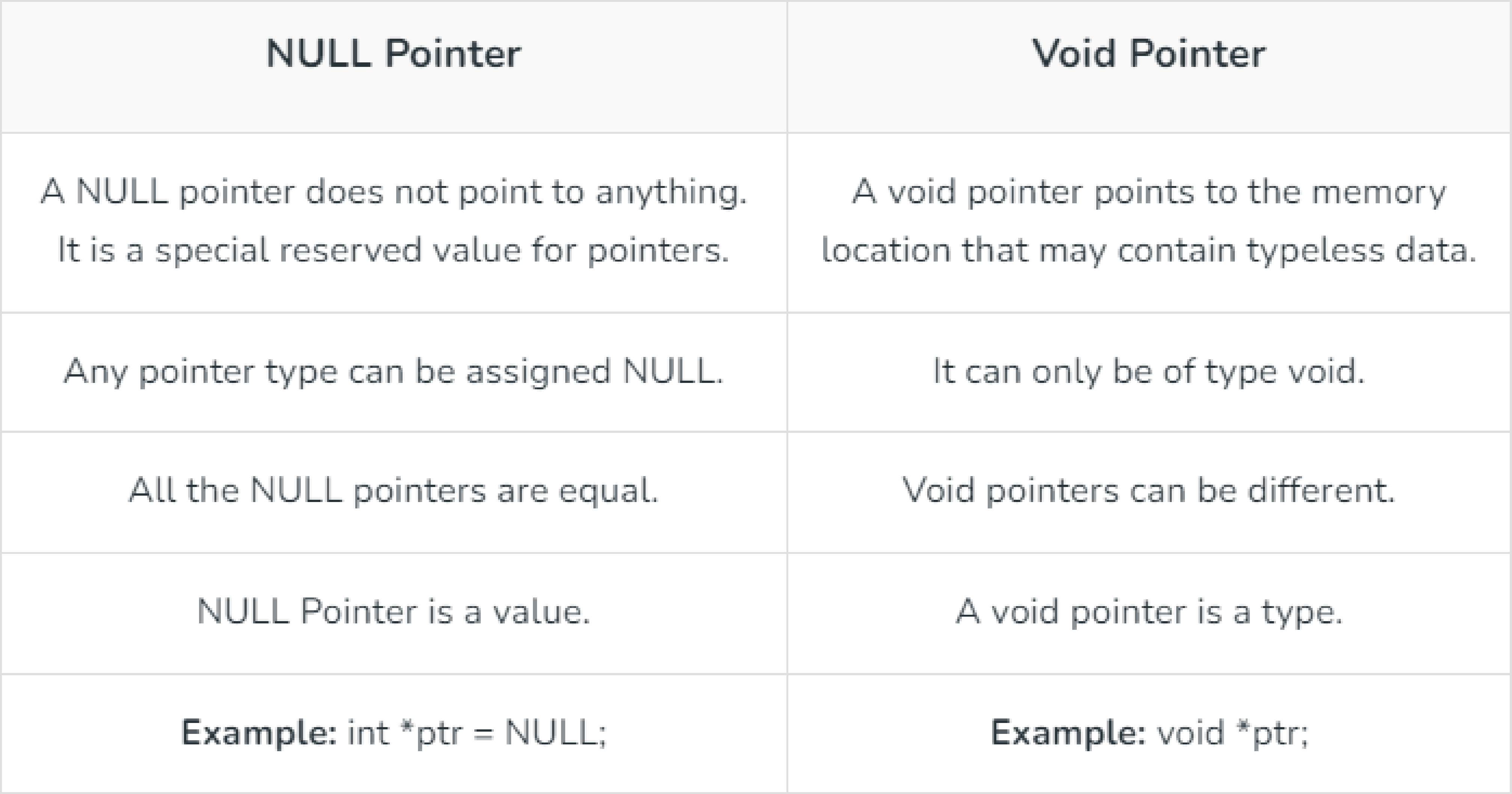


```
int a = 200;
void * ptr1 = &a;
printf("%d", *(int*) ptr1);
printf("\n%d", *(char*) ptr1);
```



```
/tmp/h299wyYiJT.o
200
-56
```

POINTER & ARRAY : NULL POINTER VS. VOID POINTER

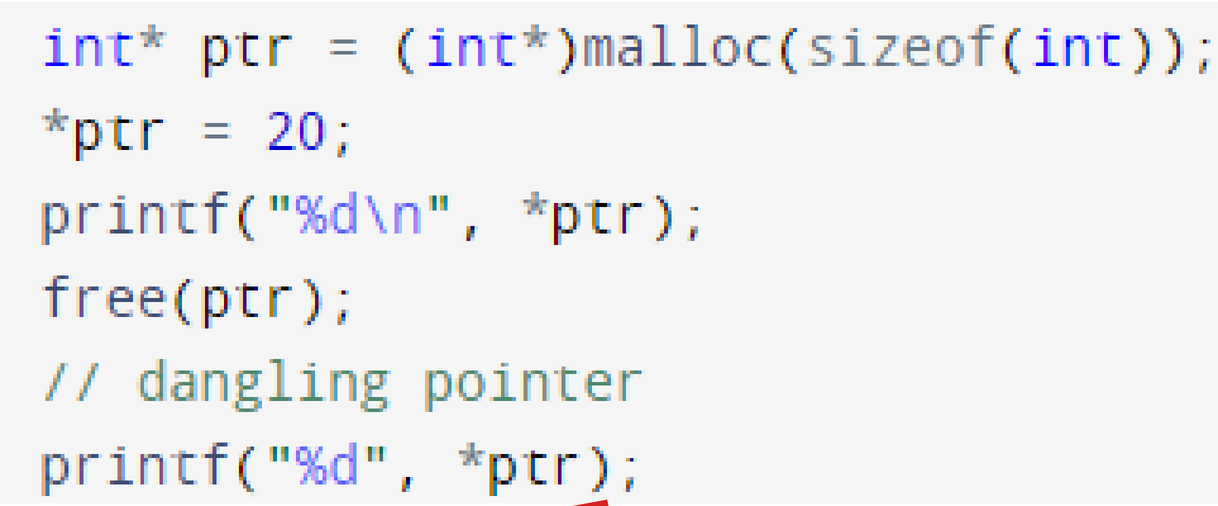


NULL Pointer	Void Pointer
A NULL pointer does not point to anything. It is a special reserved value for pointers.	A void pointer points to the memory location that may contain typeless data.
Any pointer type can be assigned NULL.	It can only be of type void.
All the NULL pointers are equal.	Void pointers can be different.
NULL Pointer is a value.	A void pointer is a type.
Example: int *ptr = NULL;	**Example:** void *ptr;

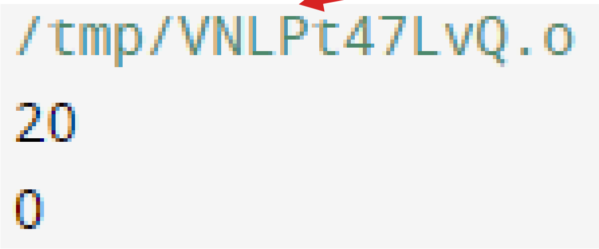
POINTER & ARRAY : DANGLING POINTER

A Pointer pointing to a memory location that has been deleted (or freed) is called a **dangling pointer**.

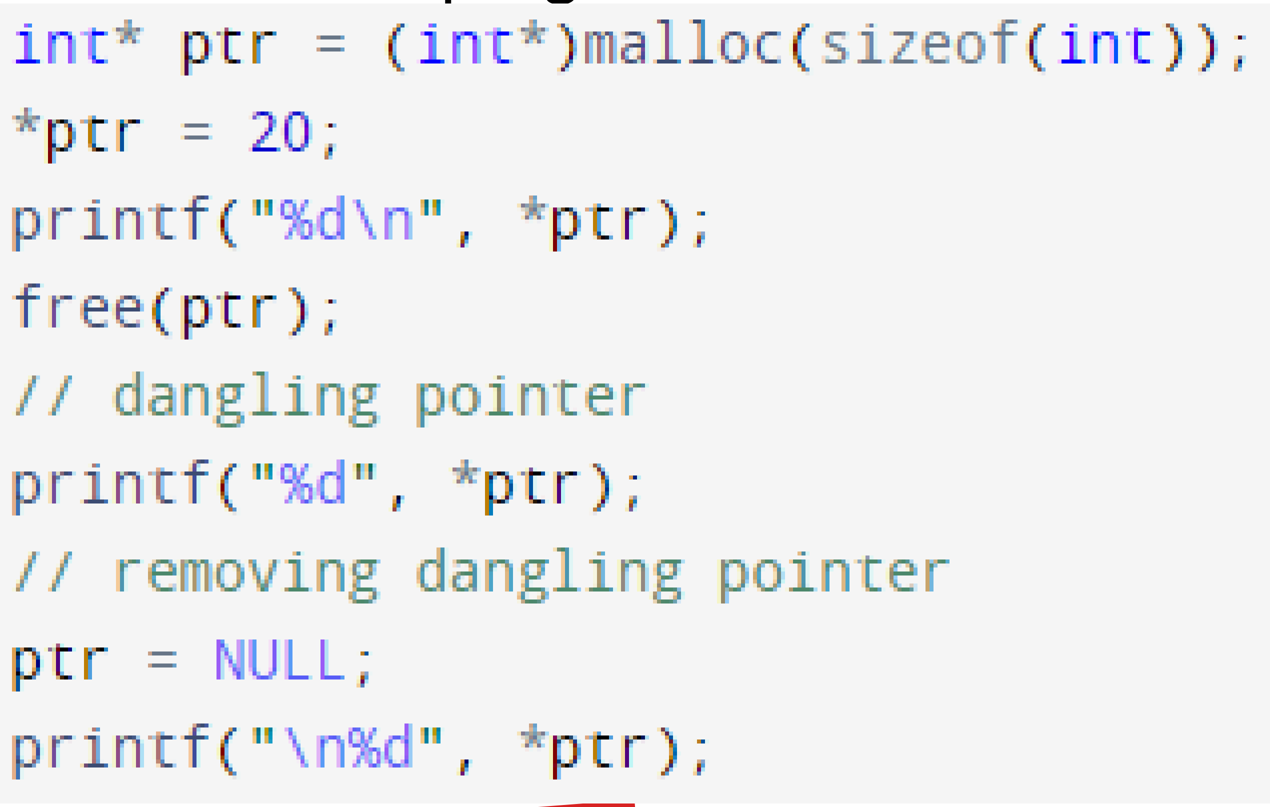
Such a situation can lead to unexpected behavior in the program and also serve as a source of bugs in C programs.



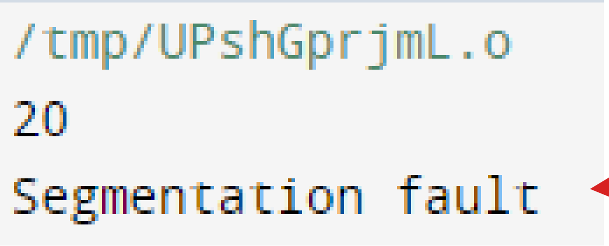
```
int* ptr = (int*)malloc(sizeof(int));
*ptr = 20;
printf("%d\n", *ptr);
free(ptr);
// dangling pointer
printf("%d", *ptr);
```



```
/tmp/VNLPt47LvQ.o
20
0
```

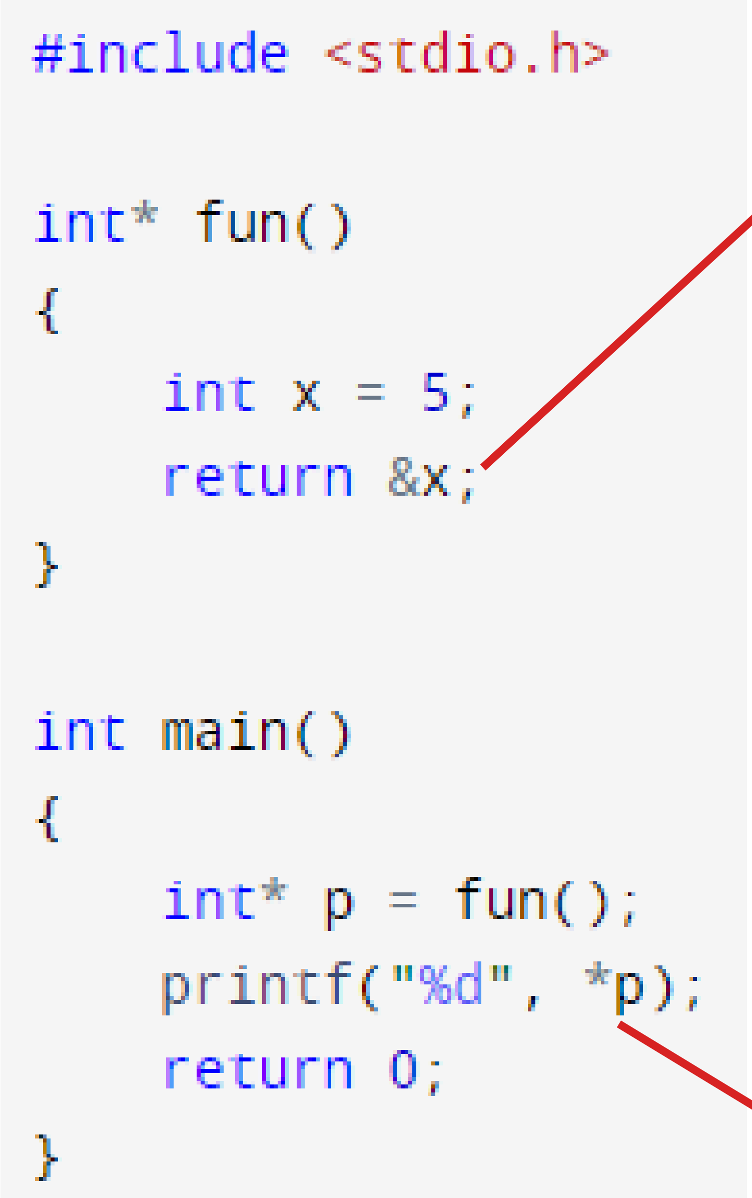


```
int* ptr = (int*)malloc(sizeof(int));
*ptr = 20;
printf("%d\n", *ptr);
free(ptr);
// dangling pointer
printf("%d", *ptr);
// removing dangling pointer
ptr = NULL;
printf("\n%d", *ptr);
```



```
/tmp/UPshGprjmL.o
20
Segmentation fault
```

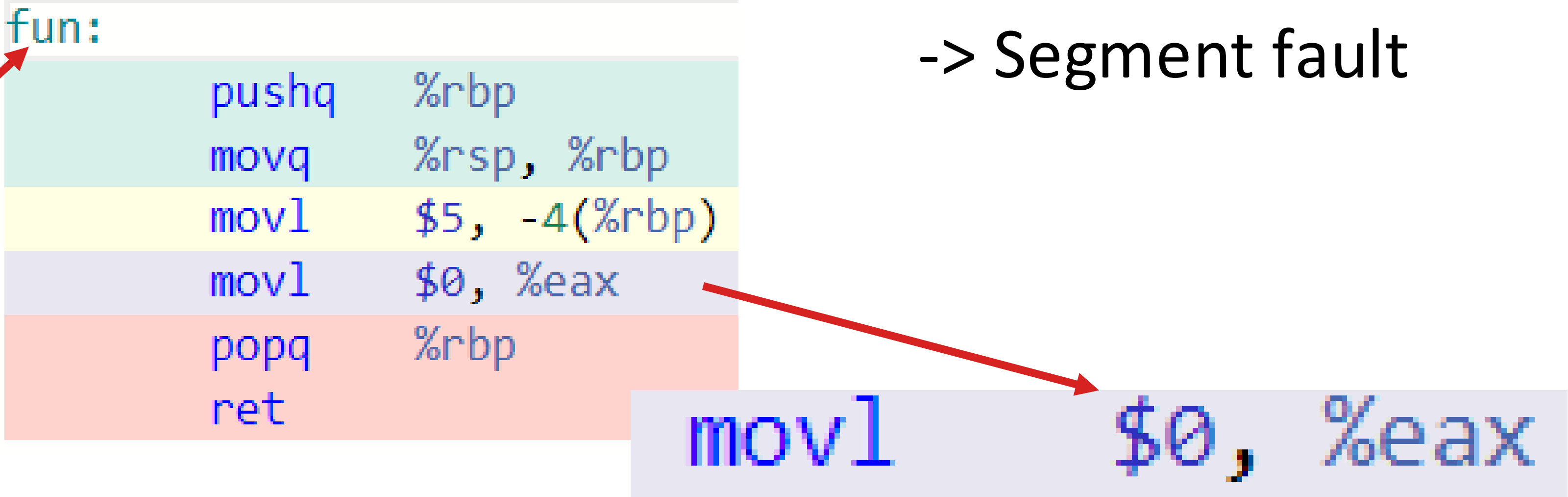
POINTER & ARRAY : DANGLING POINTER



```c
#include <stdio.h>

int* fun()
{
    int x = 5;
    return &x;
}

int main()
{
    int* p = fun();
    printf("%d", *p);
    return 0;
}
```

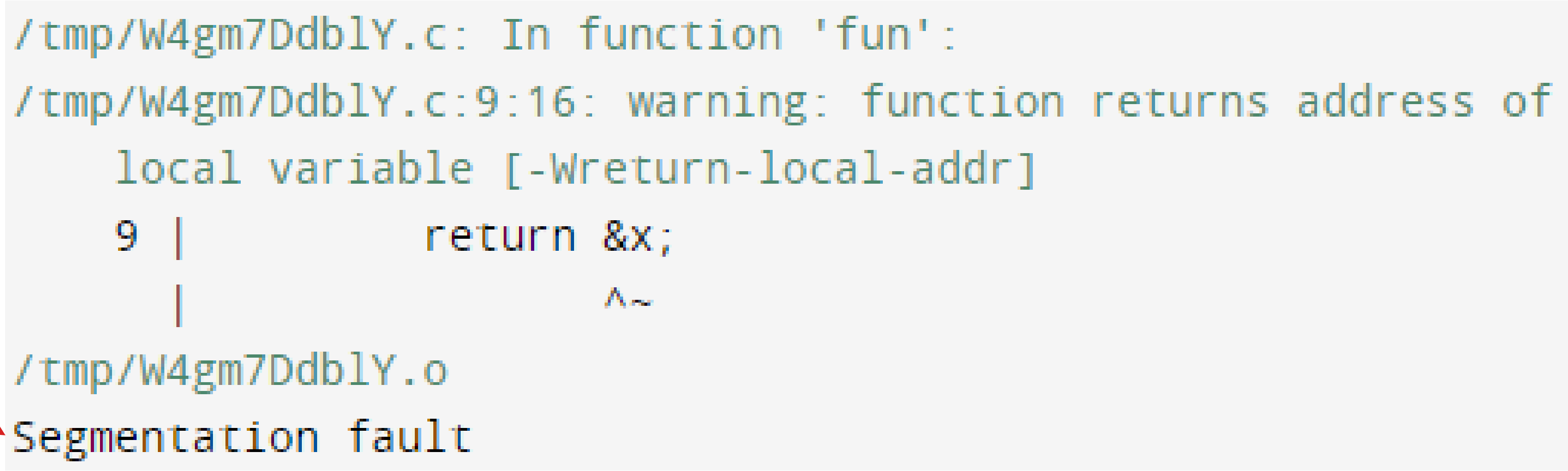
Return Address is 0 but couldn’t access to address 0

-> Segment fault



```
fun:
        pushq   %rbp
        movq    %rsp, %rbp
        movl    $5, -4(%rbp)
        movl    $0, %eax
        popq    %rbp
        ret
```

```
movl    $0, %eax
```

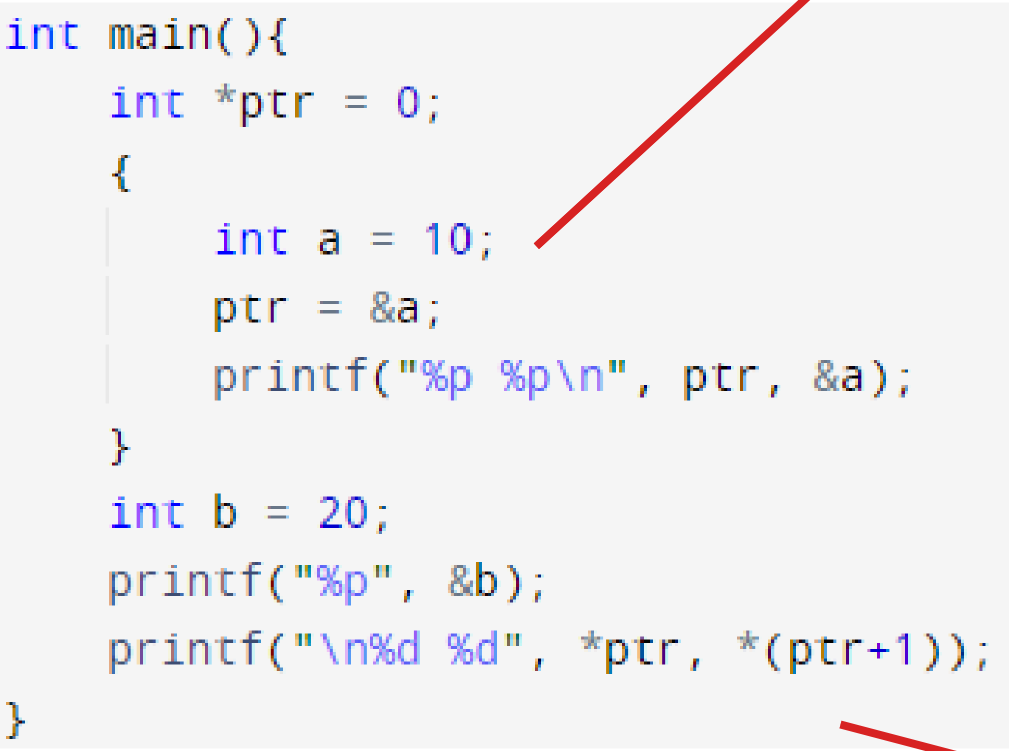


```
/tmp/W4gm7DdblY.c: In function 'fun':
/tmp/W4gm7DdblY.c:9:16: warning: function returns address of
    local variable [-Wreturn-local-addr]
    9 |         return &x;
      |                ^~
/tmp/W4gm7DdblY.o
Segmentation fault
```

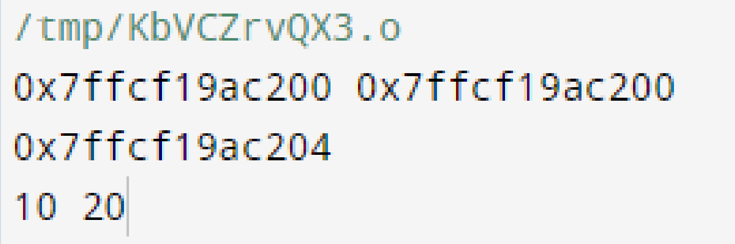
POINTER & ARRAY : DANGLING POINTER

ptr now is a dangling pointer, it can access to var a address (out of scope)

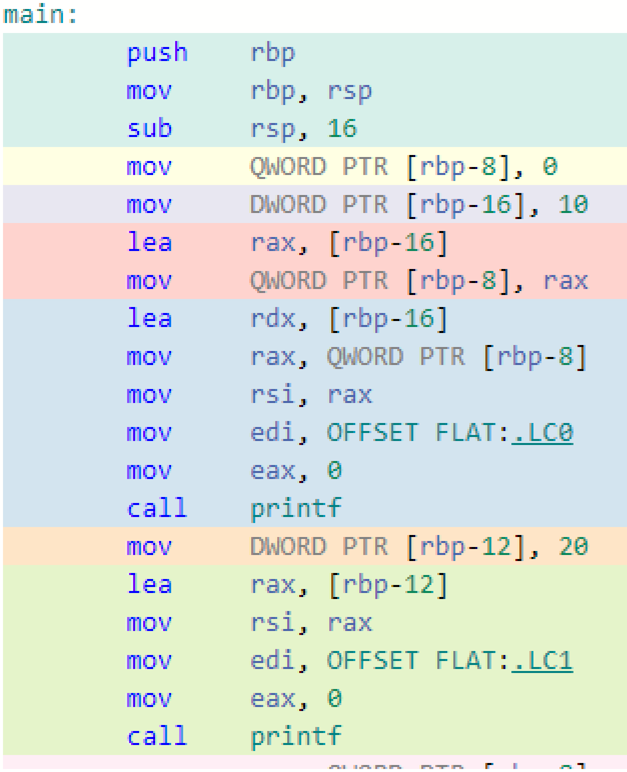
-> not raise ERROR



```c
int main(){
    int *ptr = 0;
    {
        int a = 10;
        ptr = &a;
        printf("%p %p\n", ptr, &a);
    }
    int b = 20;
    printf("%p", &b);
    printf("\n%d %d", *ptr, *(ptr+1));
}
```

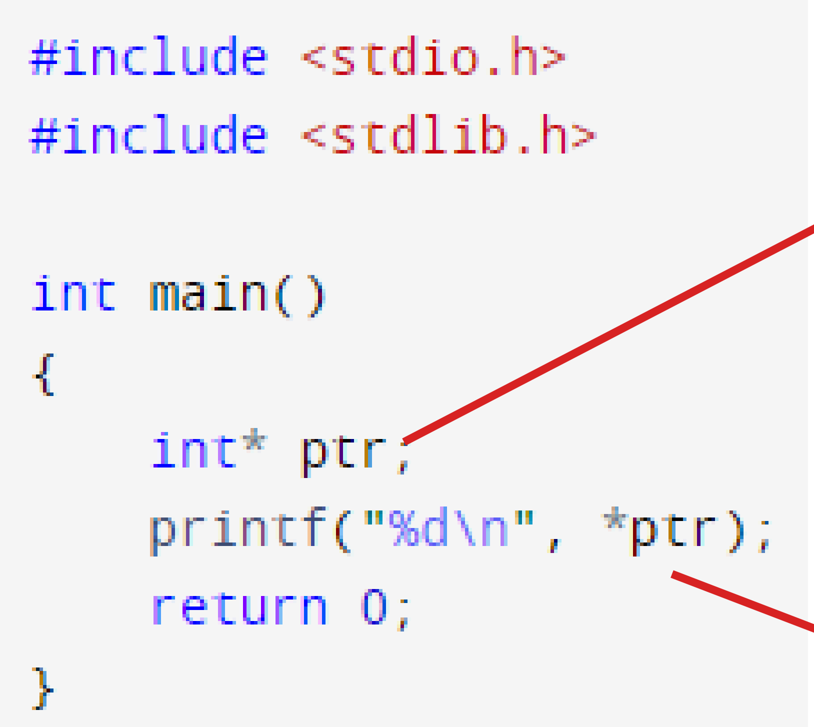


```
/tmp/KbVCZrvQX3.o
0x7ffcf19ac200 0x7ffcf19ac200
0x7ffcf19ac204
10 20
```



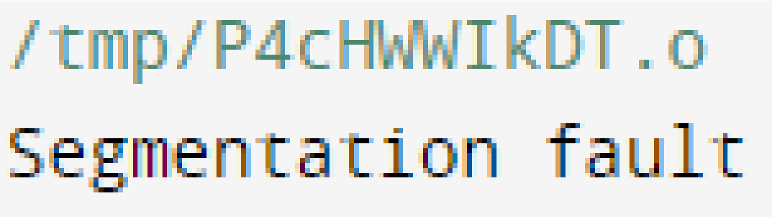
```
main:
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     QWORD PTR [rbp-8], 0
        mov     DWORD PTR [rbp-16], 10
        lea     rax, [rbp-16]
        mov     QWORD PTR [rbp-8], rax
        lea     rdx, [rbp-16]
        mov     rax, QWORD PTR [rbp-8]
        mov     rsi, rax
        mov     edi, OFFSET FLAT:.LC0
        mov     eax, 0
        call    printf
        mov     DWORD PTR [rbp-12], 20
        lea     rax, [rbp-12]
        mov     rsi, rax
        mov     edi, OFFSET FLAT:.LC1
        mov     eax, 0
        call    printf
```

POINTER & ARRAY : WILD POINTER



```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int* ptr;
    printf("%d\n", *ptr);
    return 0;
}
```

ptr now is a wild pointer, it may be initialized to a non-NULL garbage value.
-> Segment fault when trying access to that garbage value

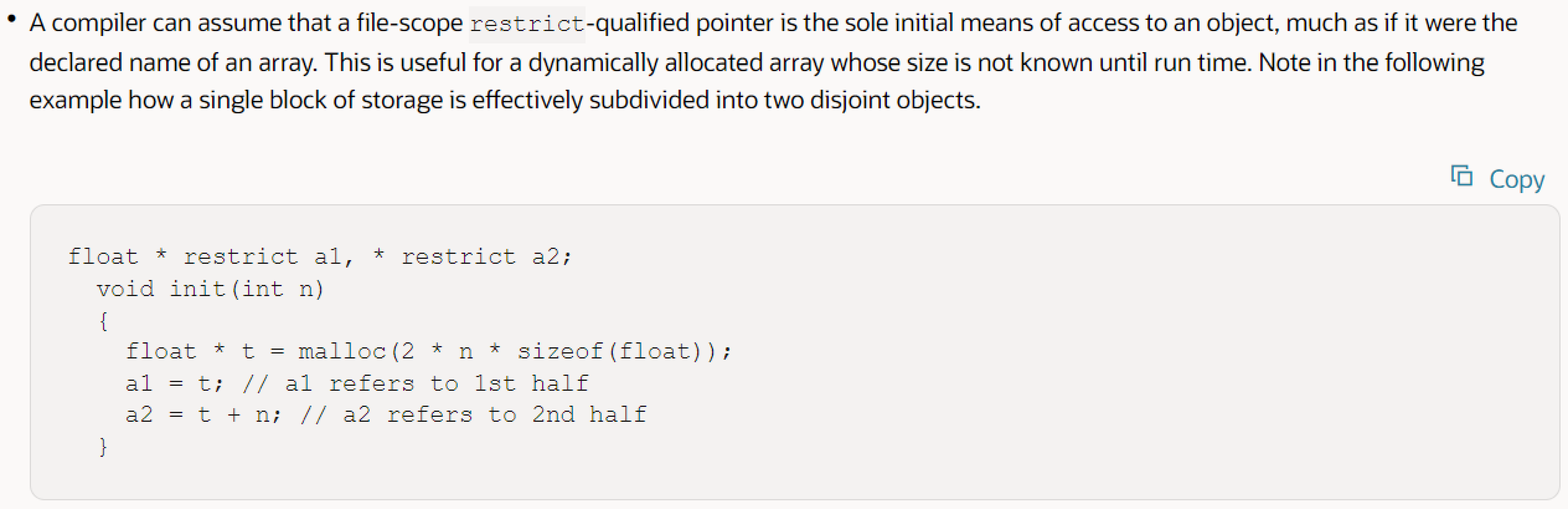


```
/tmp/P4cHWWIkDT.o
Segmentation fault
```

POINTER & ARRAY : RESTRICT QUALIFIED

Objects referenced through a **restrict-qualified** pointer have a special association with that pointer. All references to that object must directly or indirectly use the value of this pointer. In the absence of this qualifier, other pointers can alias this object. Cacheing the value in an object designated through a **restrict-qualified** pointer is safe at the beginning of the block in which the pointer is declared, because no pre-existing aliases may also be used to reference that object. The cached value must be restored to the object by the end of the block, where pre-existing aliases again become available. New aliases may be formed within the block, but these must all depend on the value of the **restrict-qualified** pointer, so that they can be identified and adjusted to refer to the cached value. For a **restrict-qualified** pointer at file scope, the block is the body of each function in the file.

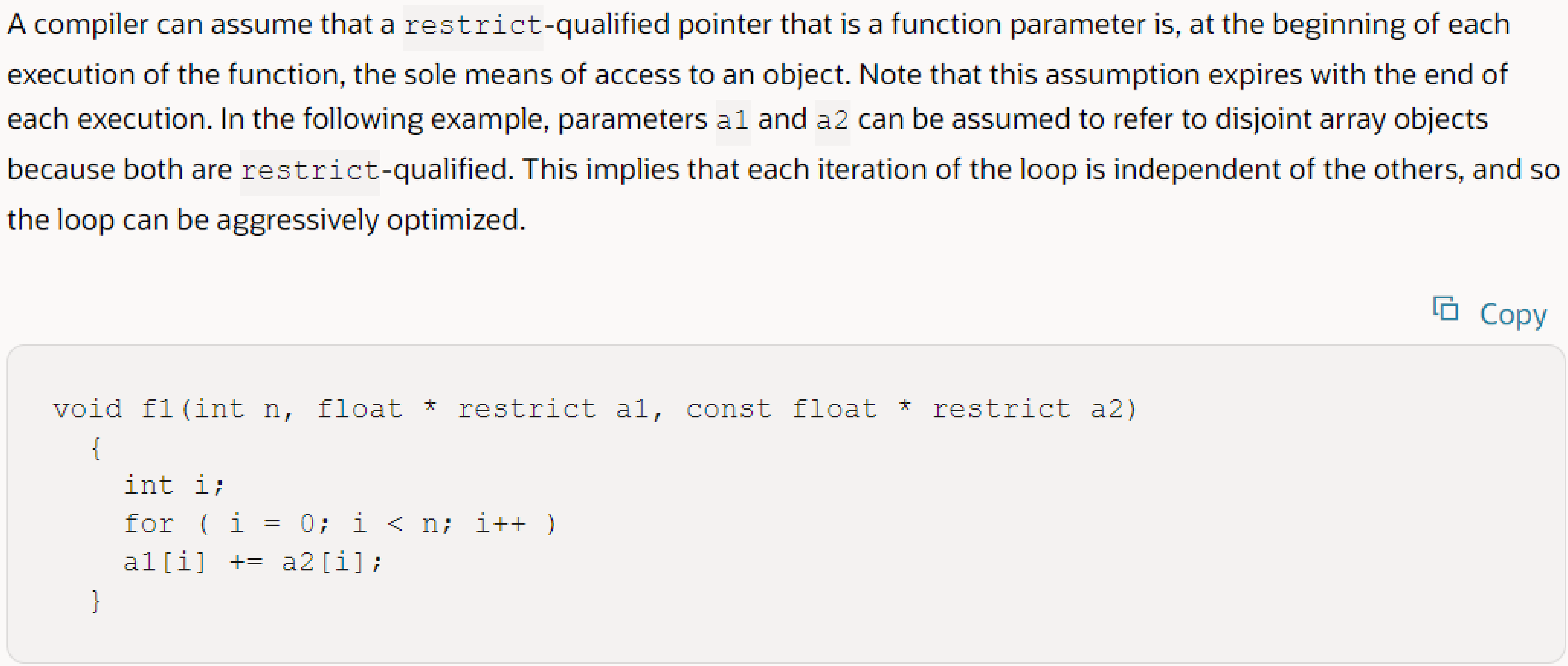
POINTER & ARRAY : RESTRICT QUALIFIED



- A compiler can assume that a file-scope `restrict`-qualified pointer is the sole initial means of access to an object, much as if it were the declared name of an array. This is useful for a dynamically allocated array whose size is not known until run time. Note in the following example how a single block of storage is effectively subdivided into two disjoint objects.

```
float * restrict a1, * restrict a2;
  void init(int n)
  {
    float * t = malloc(2 * n * sizeof(float));
    a1 = t; // a1 refers to 1st half
    a2 = t + n; // a2 refers to 2nd half
  }
```

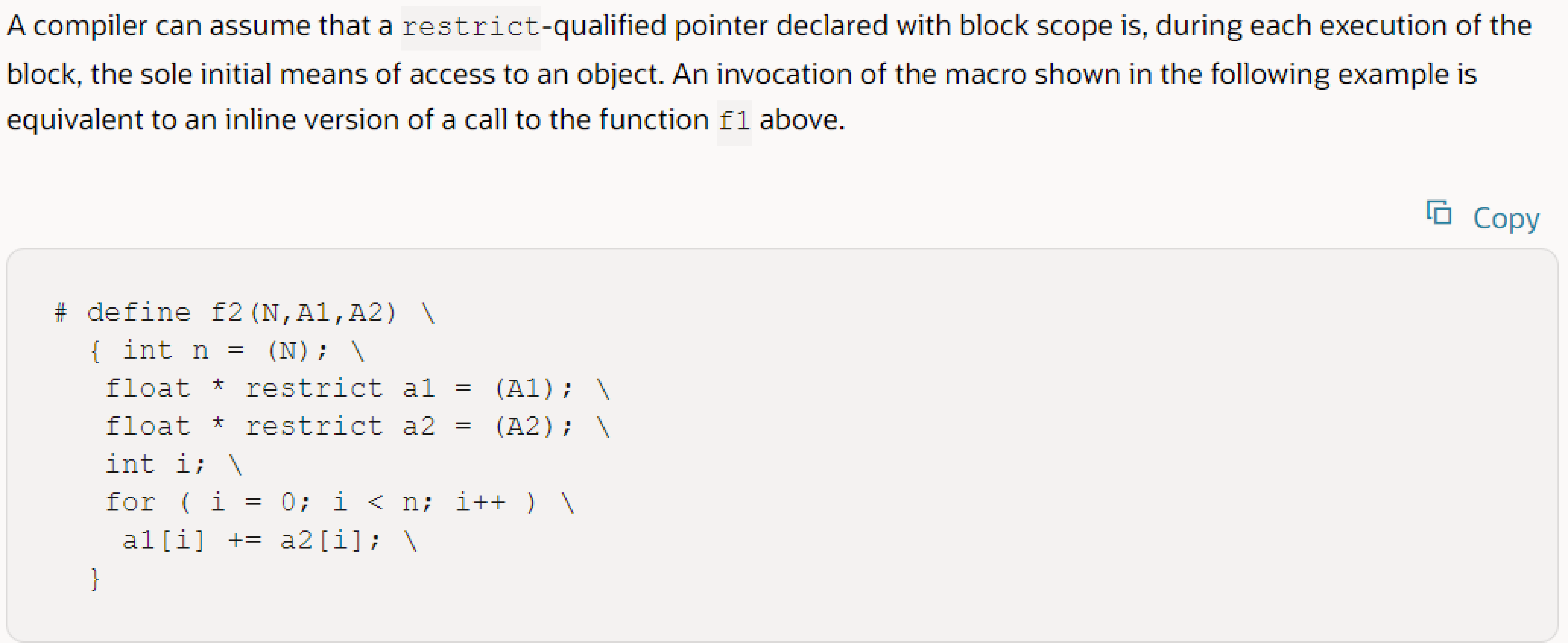
POINTER & ARRAY : RESTRICT QUALIFIED



A compiler can assume that a `restrict`-qualified pointer that is a function parameter is, at the beginning of each execution of the function, the sole means of access to an object. Note that this assumption expires with the end of each execution. In the following example, parameters `a1` and `a2` can be assumed to refer to disjoint array objects because both are `restrict`-qualified. This implies that each iteration of the loop is independent of the others, and so the loop can be aggressively optimized.

```
void f1(int n, float * restrict a1, const float * restrict a2)
 {
  int i;
  for ( i = 0; i < n; i++ )
  a1[i] += a2[i];
 }
```

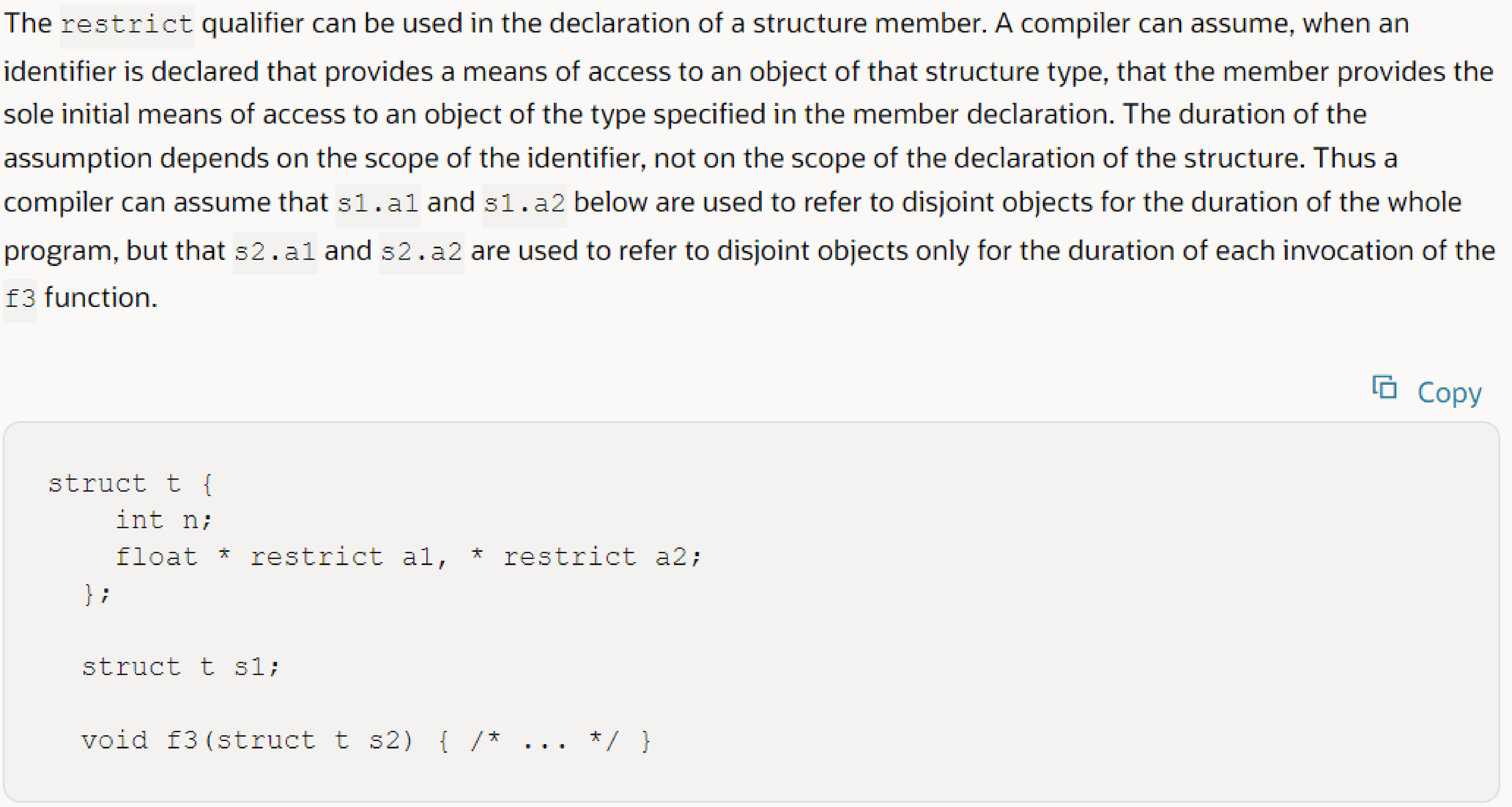
POINTER & ARRAY : RESTRICT QUALIFIED



A compiler can assume that a `restrict`-qualified pointer declared with block scope is, during each execution of the block, the sole initial means of access to an object. An invocation of the macro shown in the following example is equivalent to an inline version of a call to the function `f1` above.

```
# define f2(N,A1,A2) \
  { int n = (N); \
   float * restrict a1 = (A1); \
   float * restrict a2 = (A2); \
   int i; \
   for ( i = 0; i < n; i++ ) \
    a1[i] += a2[i]; \
  }
```

POINTER & ARRAY : RESTRICT QUALIFIED



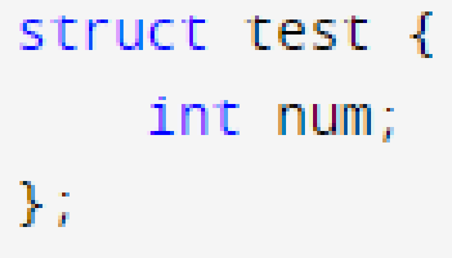
The `restrict` qualifier can be used in the declaration of a structure member. A compiler can assume, when an identifier is declared that provides a means of access to an object of that structure type, that the member provides the sole initial means of access to an object of the type specified in the member declaration. The duration of the assumption depends on the scope of the identifier, not on the scope of the declaration of the structure. Thus a compiler can assume that `s1.a1` and `s1.a2` below are used to refer to disjoint objects for the duration of the whole program, but that `s2.a1` and `s2.a2` are used to refer to disjoint objects only for the duration of each invocation of the `f3` function.

```
struct t {
    int n;
    float * restrict a1, * restrict a2;
  };

  struct t s1;

  void f3(struct t s2) { /* ... */ }
```

POINTER & ARRAY : POINTER ARITHMETIC

Pointer operator:

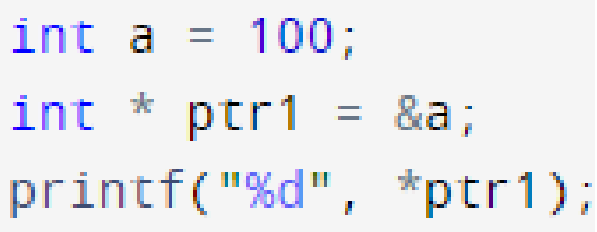
- **Operator *, ->, &**
- Increment and decrement
- Addition and subtraction
- Comparison
- Assignment
- Array



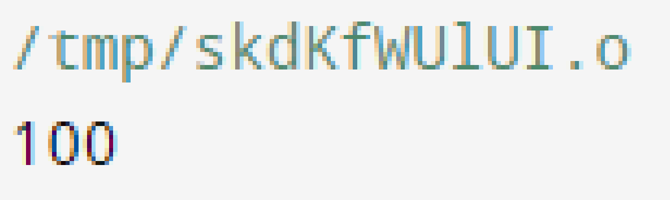
```
struct test {
    int num;
};
```

Operator *: using to get the value pointed to

Operator &: using to get the address of variable

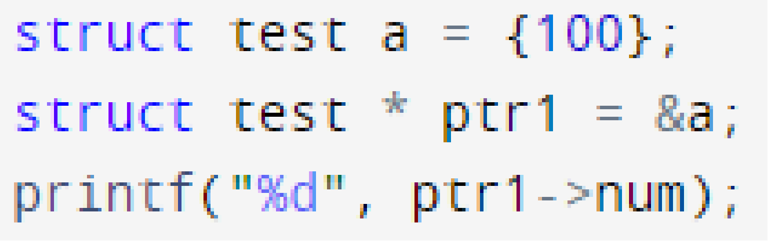


```
int a = 100;
int * ptr1 = &a;
printf("%d", *ptr1);
```

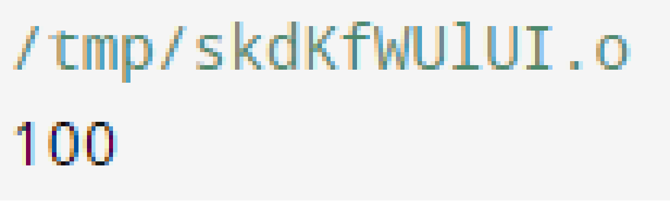


```
/tmp/skdKfWUlUI.o
100
```

Operator ->: using to get the value pointed to but using only with struct pointer



```
struct test a = {100};
struct test * ptr1 = &a;
printf("%d", ptr1->num);
```



```
/tmp/skdKfWUlUI.o
100
```

POINTER & ARRAY : POINTER ARITHMETIC

Pointer operator:

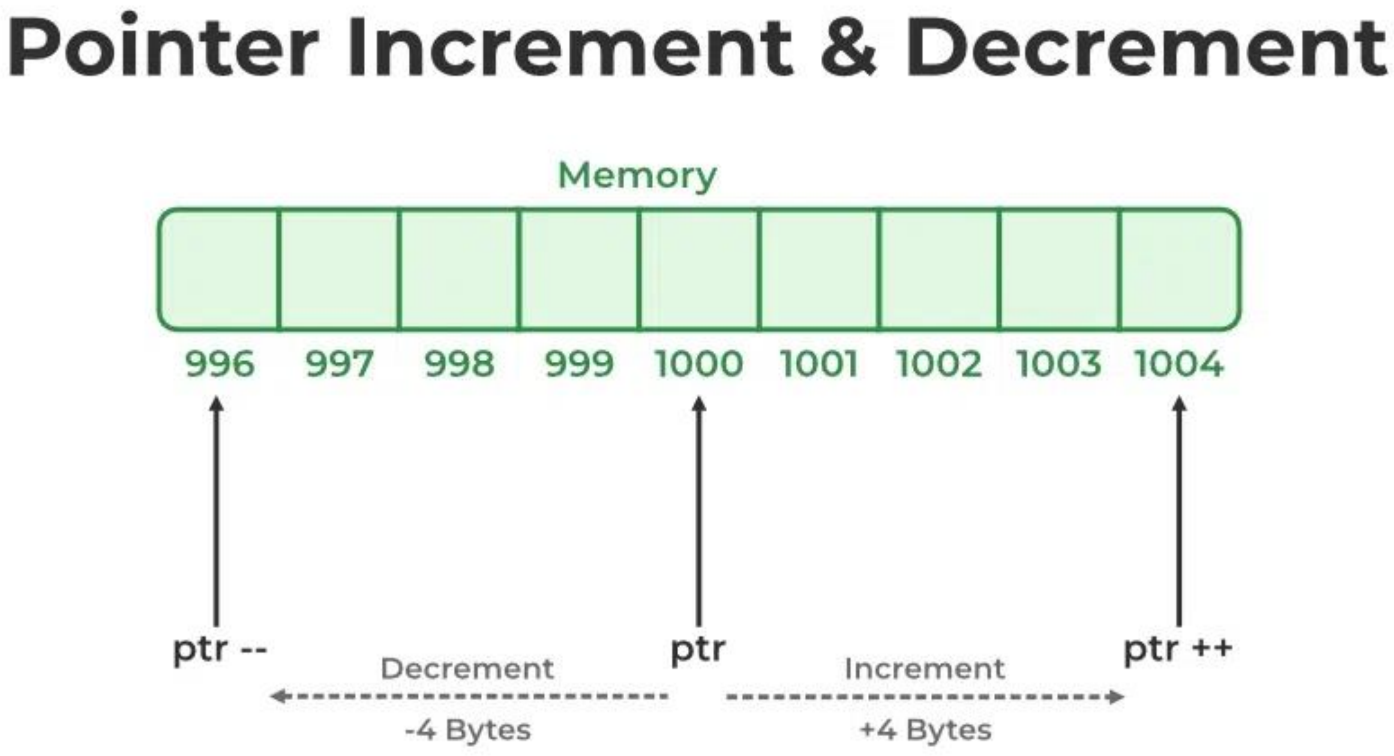
- Operator *, ->, &
- **Increment and decrement**
- Addition and subtraction
- Comparison
- Assignment
- Array

Prefix ++

Prefix –

Postfix ++

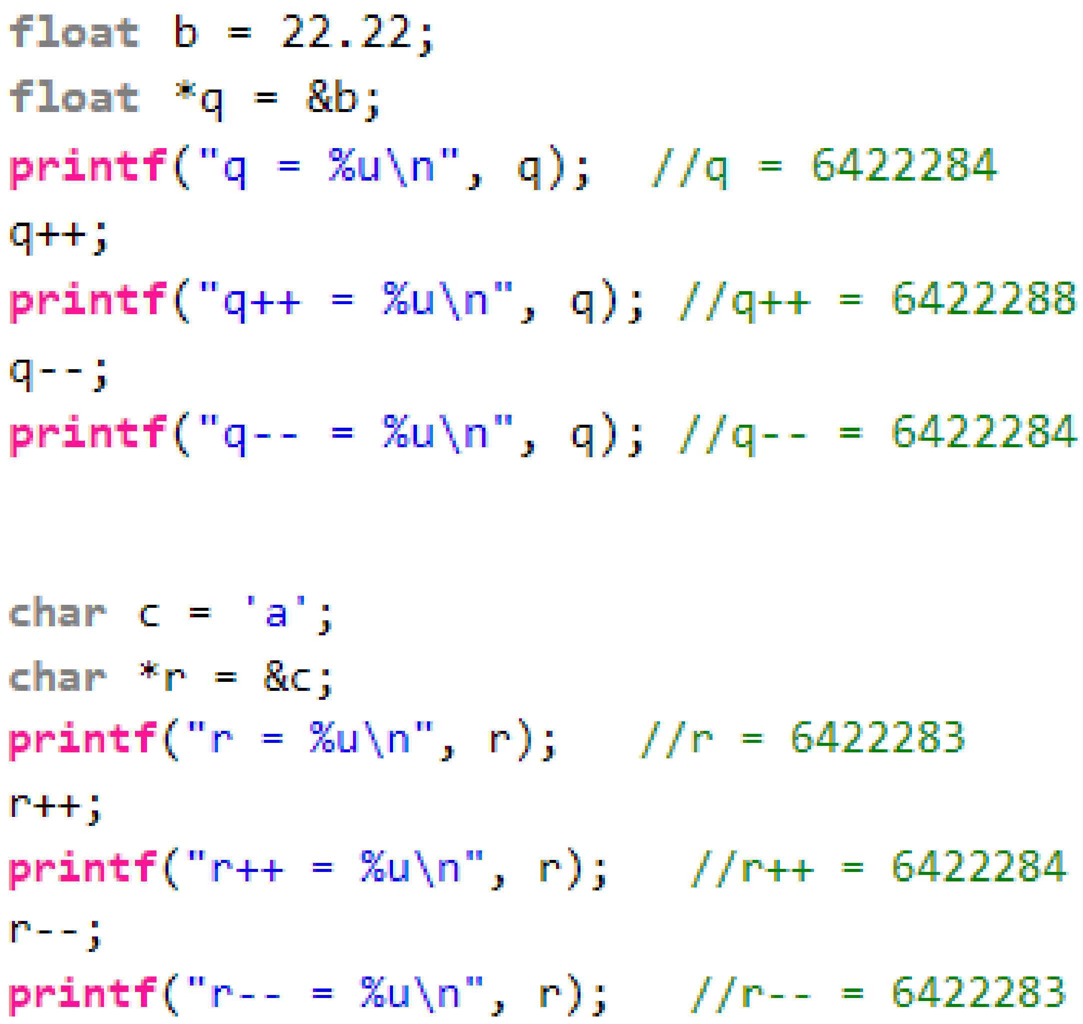
Postfix --



POINTER & ARRAY : POINTER ARITHMETIC

Pointer operator:

- Operator *, ->, &
- **Increment and decrement**
- Addition and subtraction
- Comparison
- Assignment
- Array



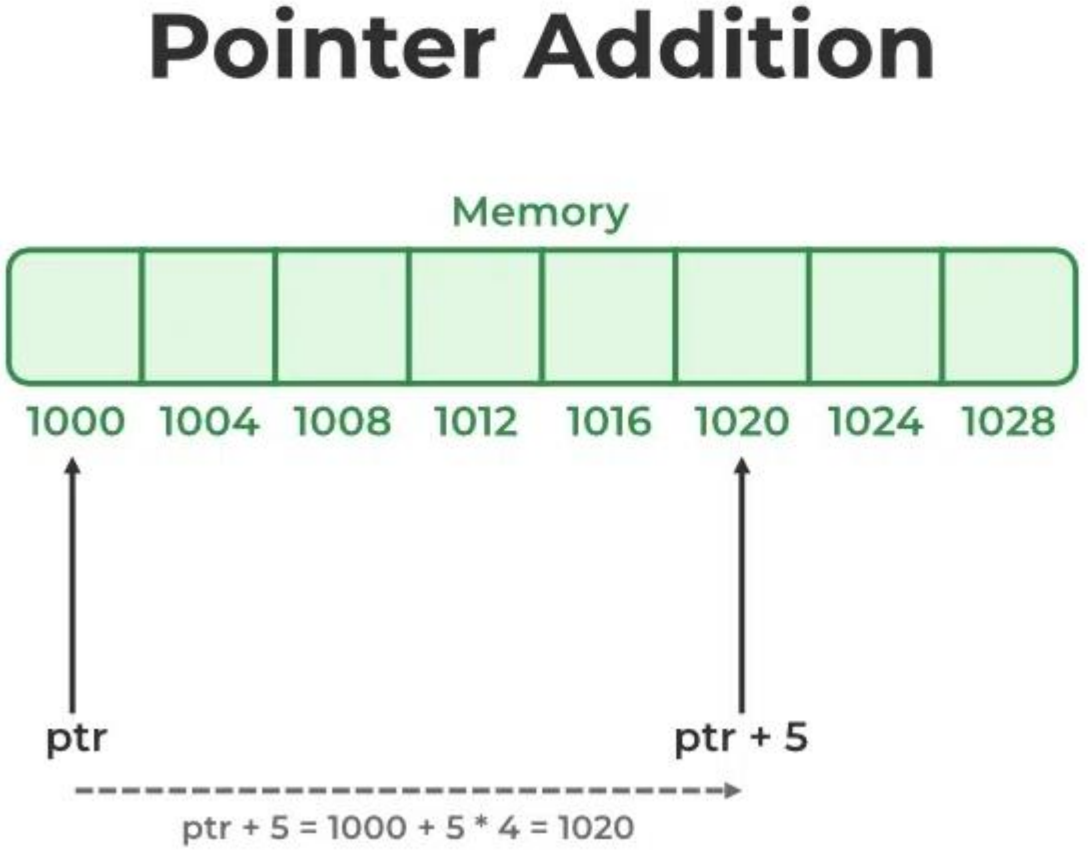
```
float b = 22.22;
float *q = &b;
printf("q = %u\n", q);  //q = 6422284
q++;
printf("q++ = %u\n", q); //q++ = 6422288
q--;
printf("q-- = %u\n", q); //q-- = 6422284


char c = 'a';
char *r = &c;
printf("r = %u\n", r);   //r = 6422283
r++;
printf("r++ = %u\n", r);   //r++ = 6422284
r--;
printf("r-- = %u\n", r);   //r-- = 6422283
```

POINTER & ARRAY : POINTER ARITHMETIC

Pointer operator:

- Operator *, ->, &
- Increment and decrement
- **Addition and subtraction**
- Comparison
- Assignment
- Array

int pointer:

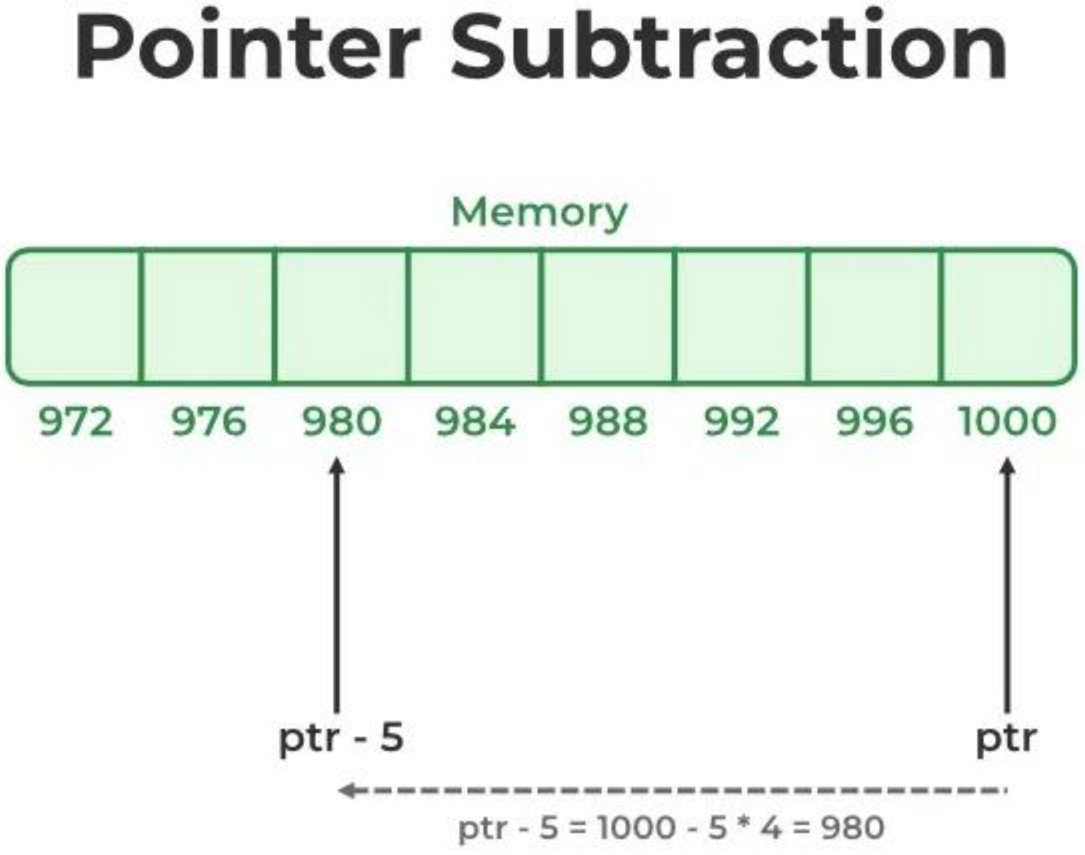


POINTER & ARRAY : POINTER ARITHMETIC

Pointer operator:

- Operator *, ->, &
- Increment and decrement
- **Addition and subtraction**
- Comparison
- Assignment
- Array

int pointer:

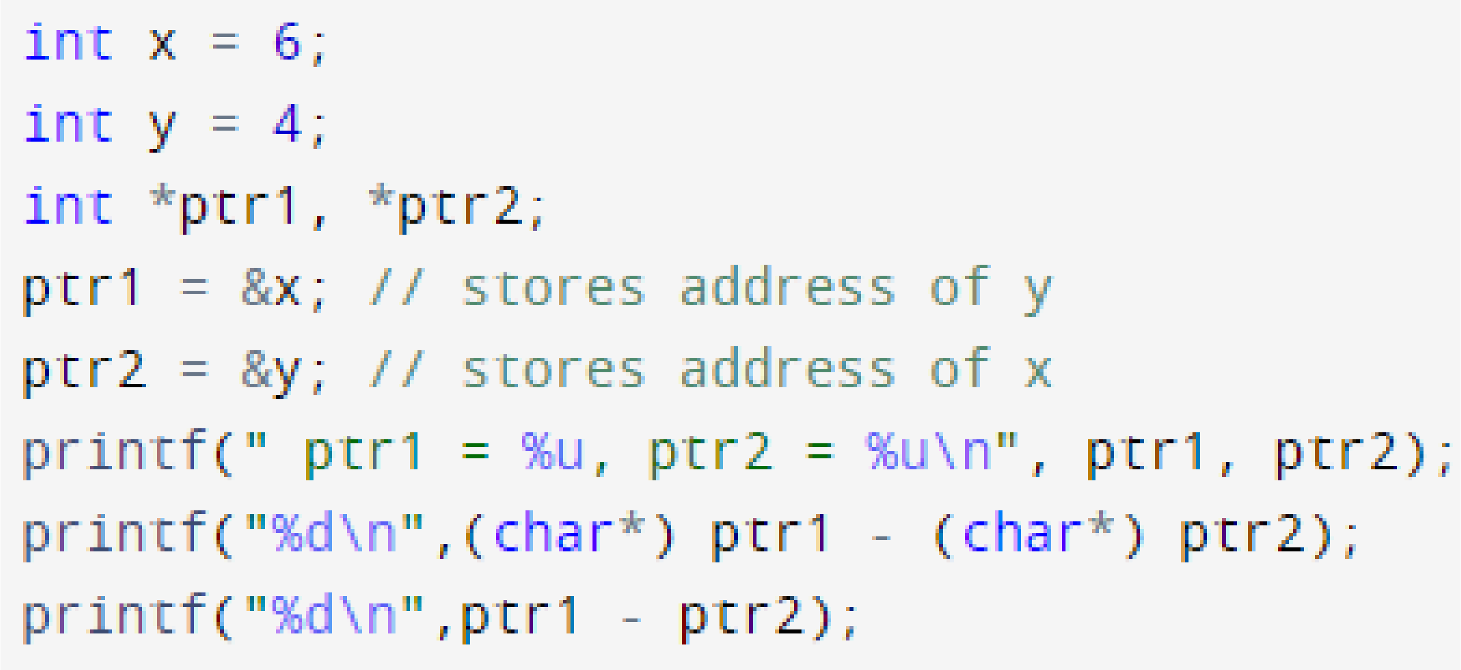


POINTER & ARRAY : POINTER ARITHMETIC

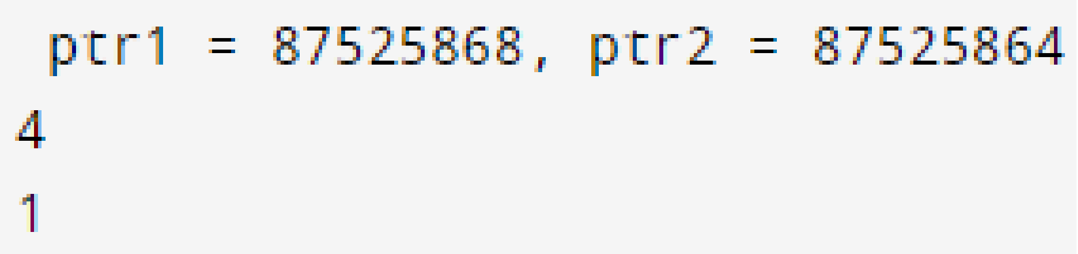
Pointer operator:

- Operator *, ->, &
- Increment and decrement
- **Addition and subtraction**
- Comparison
- Assignment
- Array

Subtraction of 2 pointer:



```c
int x = 6;
int y = 4;
int *ptr1, *ptr2;
ptr1 = &x; // stores address of y
ptr2 = &y; // stores address of x
printf(" ptr1 = %u, ptr2 = %u\n", ptr1, ptr2);
printf("%d\n",(char*) ptr1 - (char*) ptr2);
printf("%d\n",ptr1 - ptr2);
```

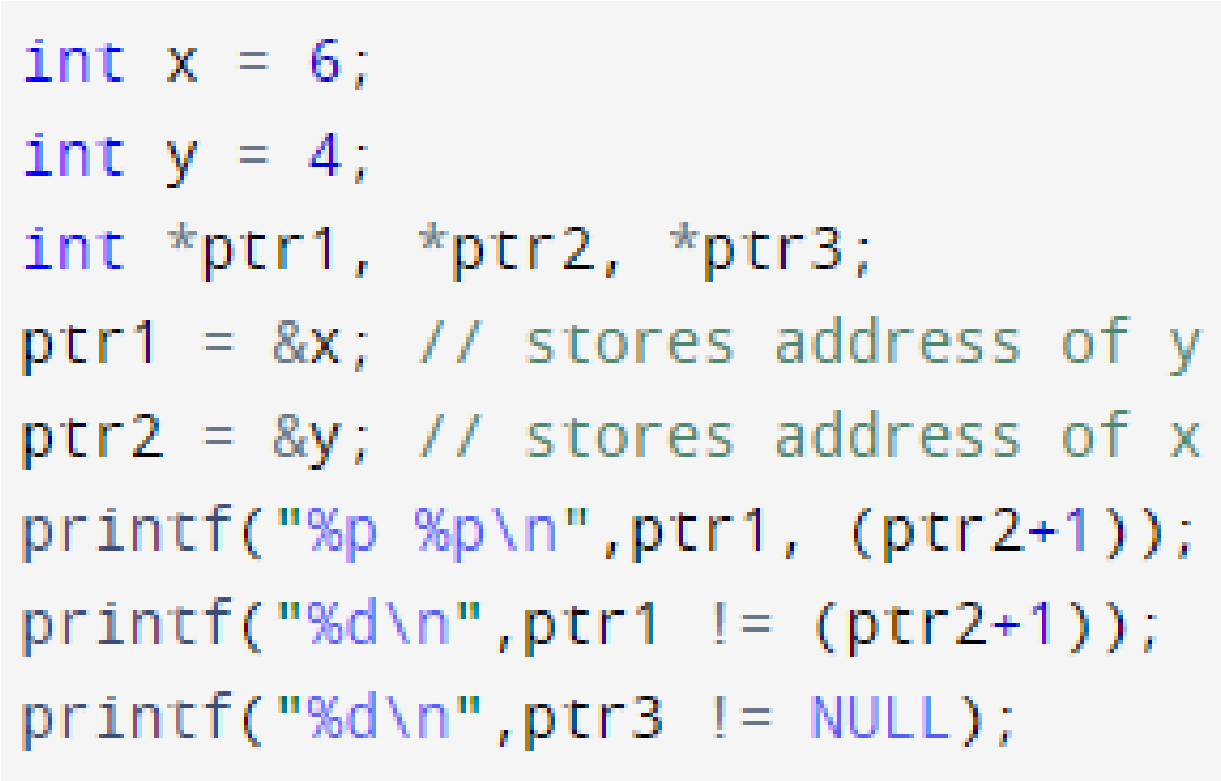


```
 ptr1 = 87525868, ptr2 = 87525864
4
1
```

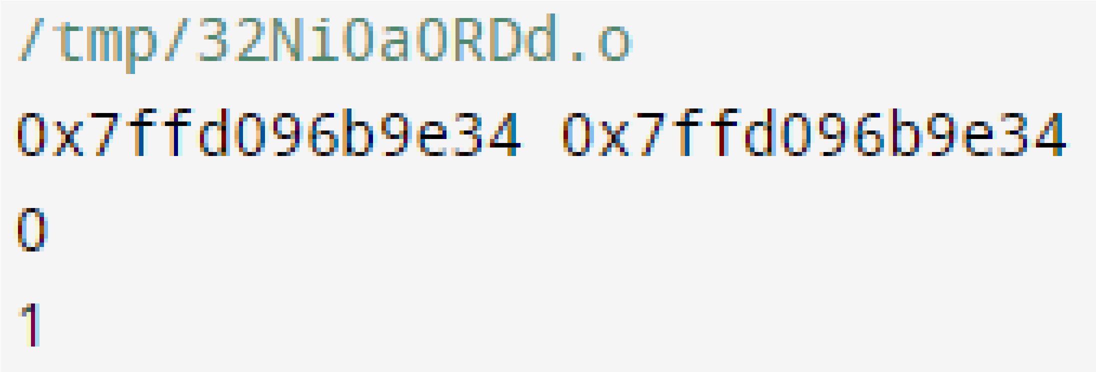
POINTER & ARRAY : POINTER ARITHMETIC

Pointer operator:

- Operator *, ->, &
- Increment and decrement
- Addition and subtraction
- **Comparison**
- Assignment
- Array



```
int x = 6;
int y = 4;
int *ptr1, *ptr2, *ptr3;
ptr1 = &x; // stores address of y
ptr2 = &y; // stores address of x
printf("%p %p\n",ptr1, (ptr2+1));
printf("%d\n",ptr1 != (ptr2+1));
printf("%d\n",ptr3 != NULL);
```



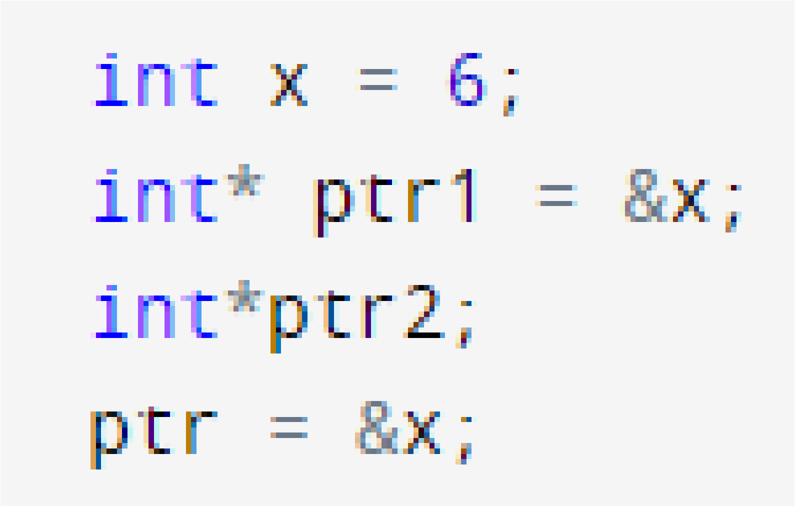
```
/tmp/32Ni0a0RDd.o
0x7ffd096b9e34 0x7ffd096b9e34
0
1
```

POINTER & ARRAY : POINTER ARITHMETIC

Pointer operator:

- Operator *, ->, &
- Increment and decrement
- Addition and subtraction
- Comparison
- **Assignment**
- Array

Assignment: 2 types:



```
int x = 6;
int* ptr1 = &x;
int*ptr2;
ptr = &x;
```

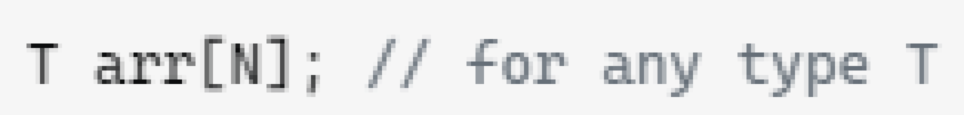
POINTER & ARRAY : POINTER ARITHMETIC

Pointer operator:

- Operator *, ->, &
- Increment and decrement
- Addition and subtraction
- Comparison
- Assignment
- **Array**

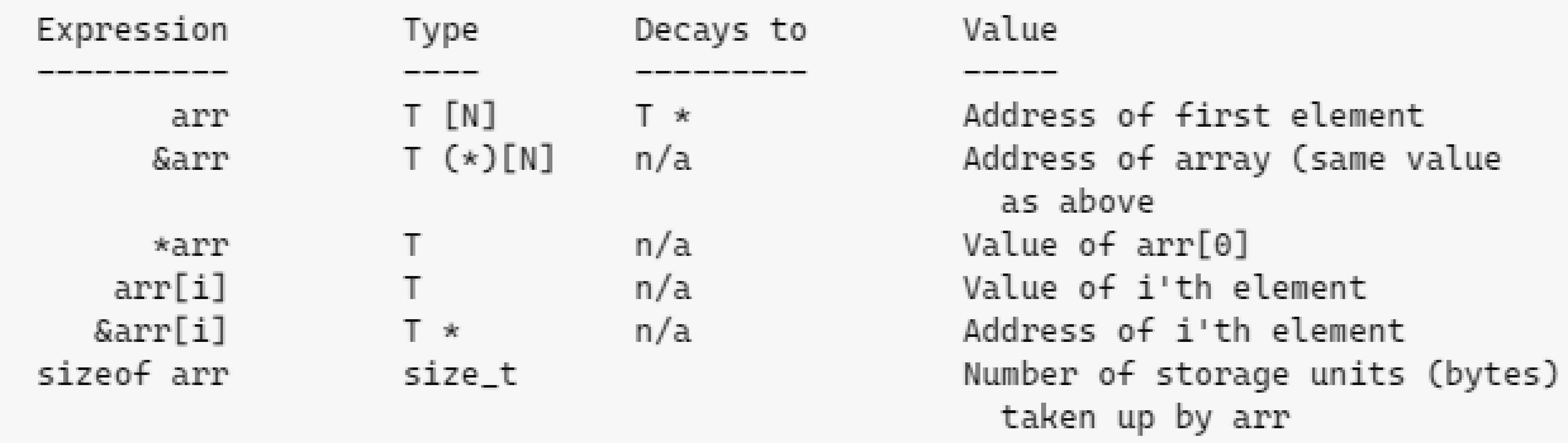
Array: name can use same as const pointer

With:



```
T arr[N]; // for any type T
```

That:

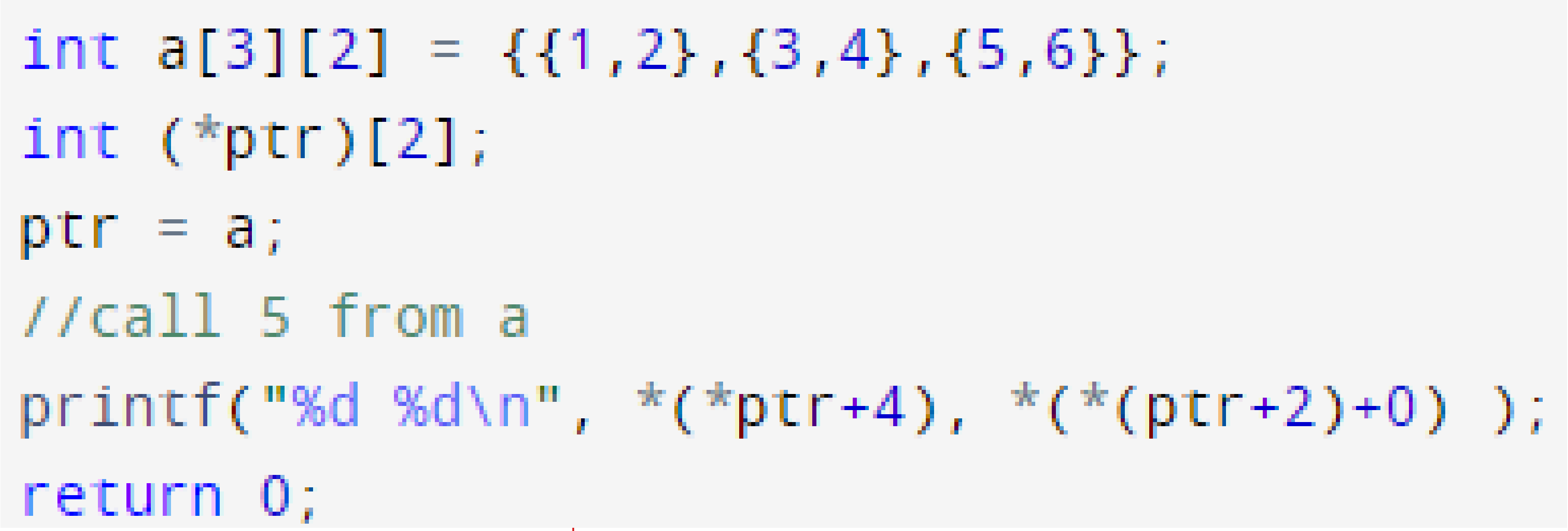


```
Expression         Type        Decays to        Value
----------         ----        ---------        -----
       arr         T [N]       T *              Address of first element
      &arr         T (*)[N]    n/a              Address of array (same value
                                                  as above
      *arr         T           n/a              Value of arr[0]
    arr[i]         T           n/a              Value of i'th element
   &arr[i]         T *         n/a              Address of i'th element
sizeof arr         size_t                       Number of storage units (bytes)
                                                  taken up by arr
```

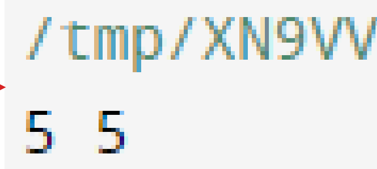
POINTER & ARRAY : POINTER ARITHMETIC

Pointer operator:

- Operator *, ->, &
- Increment and decrement
- Addition and subtraction
- Comparison
- Assignment
- **Array**



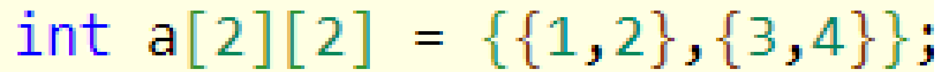
```
int a[3][2] = {{1,2},{3,4},{5,6}};
int (*ptr)[2];
ptr = a;
//call 5 from a
printf("%d %d\n", *(*ptr+4), *(*(ptr+2)+0) );
return 0;
```



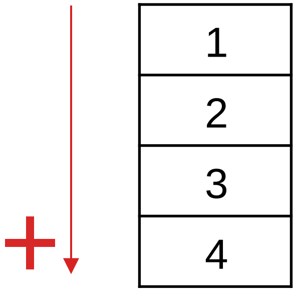
```
/tmp/XN9VV
5 5
```

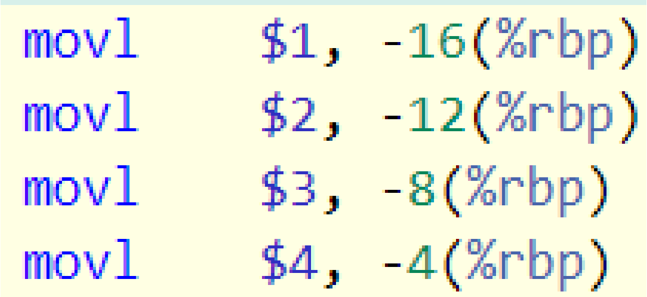
Multi-dimensional Array:

Address of array



```
int a[2][2] = {{1,2},{3,4}};
```



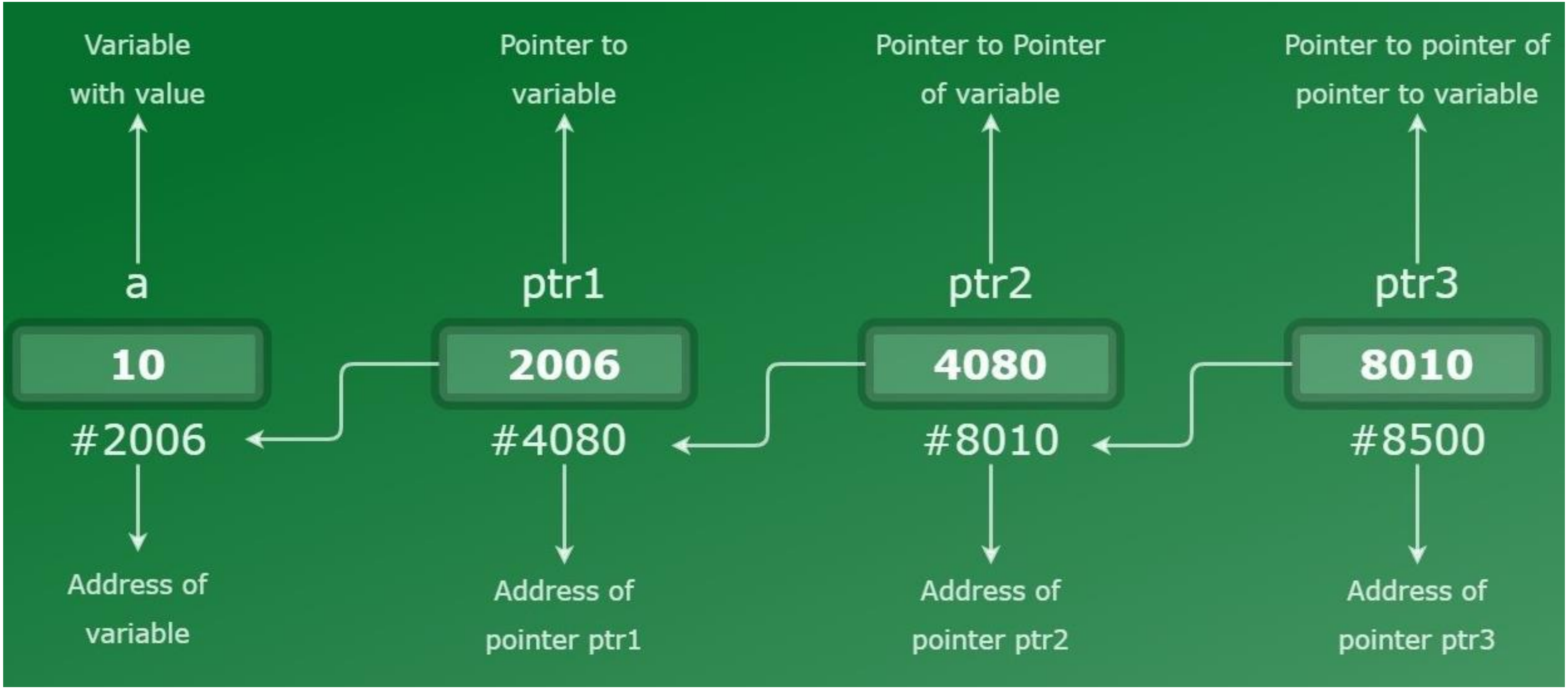


```
movl    $1, -16(%rbp)
movl    $2, -12(%rbp)
movl    $3, -8(%rbp)
movl    $4, -4(%rbp)
```

POINTER & ARRAY : POINTER TO POINTER

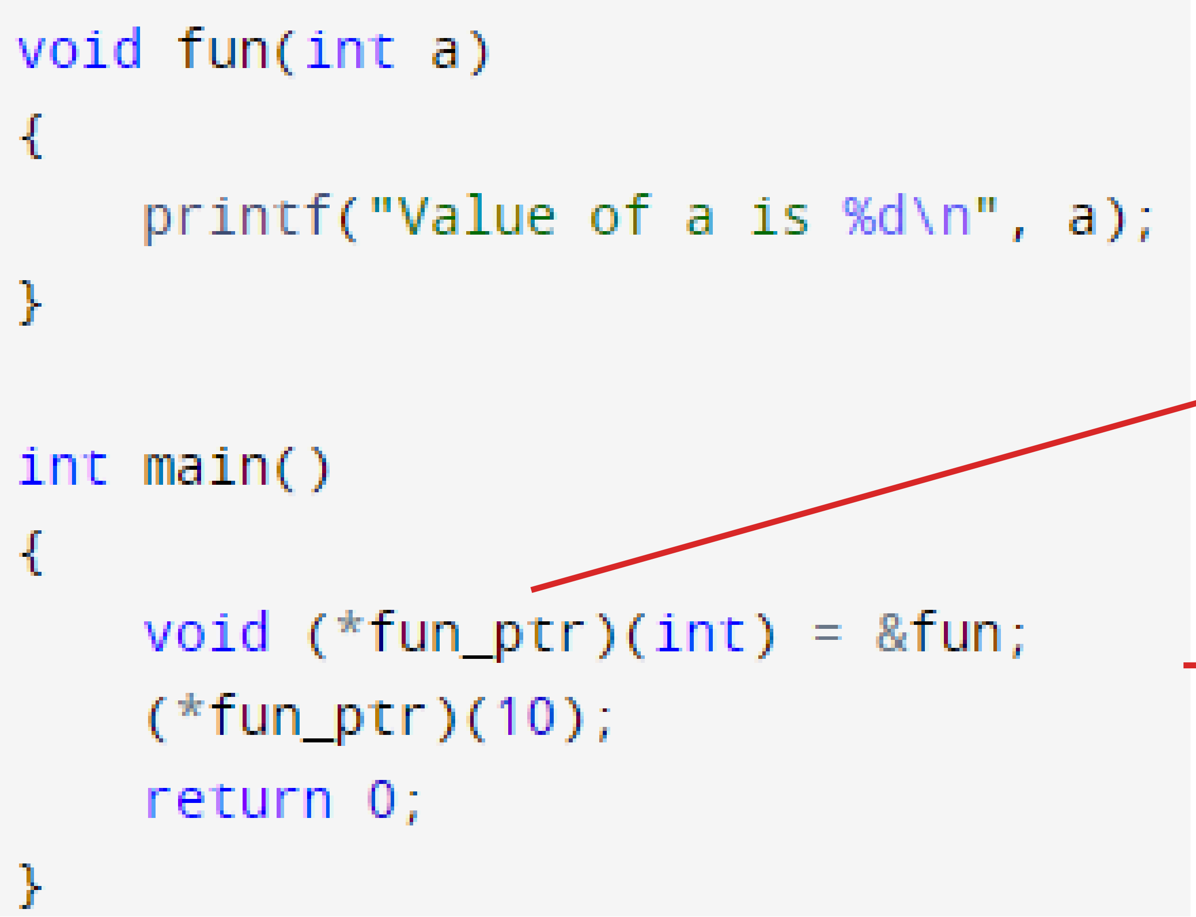
Multilevel pointer in C:

```
pointer_type *** pointer_name;
```



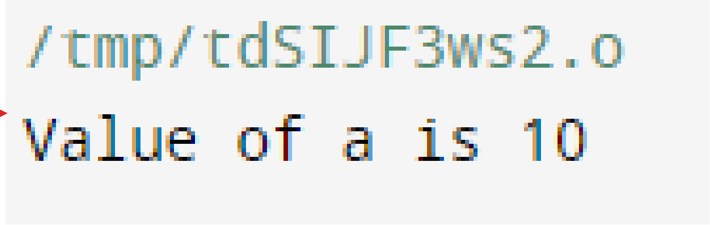
POINTER & FUNCTION: FUNCTION POINTER

In C, like normal data pointers (int *, char *, etc), we can have pointers to functions. You can using this pointer to call function.



```
void fun(int a)
{
    printf("Value of a is %d\n", a);
}

int main()
{
    void (*fun_ptr)(int) = &fun;
    (*fun_ptr)(10);
    return 0;
}
```

Function pointer (point to function fun)

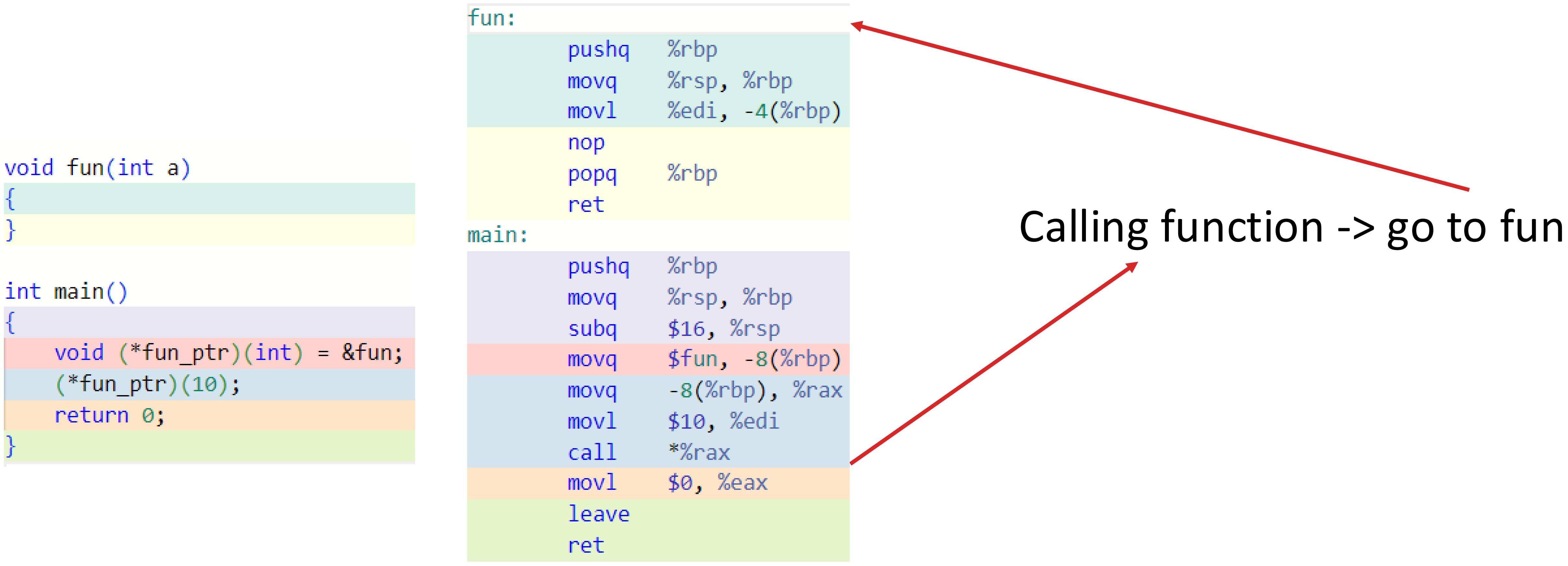


```
/tmp/tdSIJF3ws2.o
Value of a is 10
```

POINTER & FUNCTION: FUNCTION POINTER

Fact:

1. Unlike normal pointers, a function pointer points to code, not data. Typically a function pointer stores the start of executable code.



```c
void fun(int a)
{
}

int main()
{
    void (*fun_ptr)(int) = &fun;
    (*fun_ptr)(10);
    return 0;
}
```

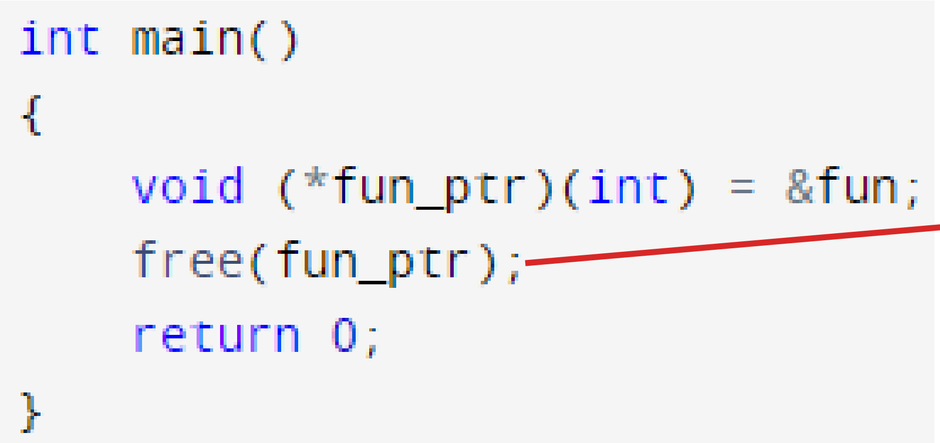
```
fun:
        pushq   %rbp
        movq    %rsp, %rbp
        movl    %edi, -4(%rbp)
        nop
        popq    %rbp
        ret
main:
        pushq   %rbp
        movq    %rsp, %rbp
        subq    $16, %rsp
        movq    $fun, -8(%rbp)
        movq    -8(%rbp), %rax
        movl    $10, %edi
        call    *%rax
        movl    $0, %eax
        leave
        ret
```

Calling function -> go to fun

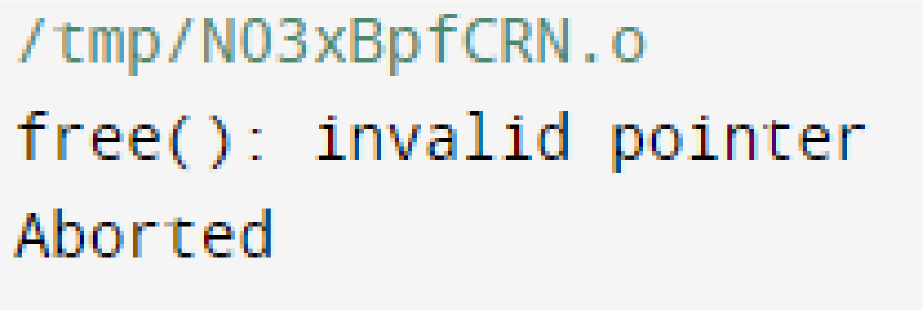
POINTER & FUNCTION: FUNCTION POINTER

Fact:

2. we do not allocate de-allocate memory using function pointers



```c
int main()
{
    void (*fun_ptr)(int) = &fun;
    free(fun_ptr);
    return 0;
}
```

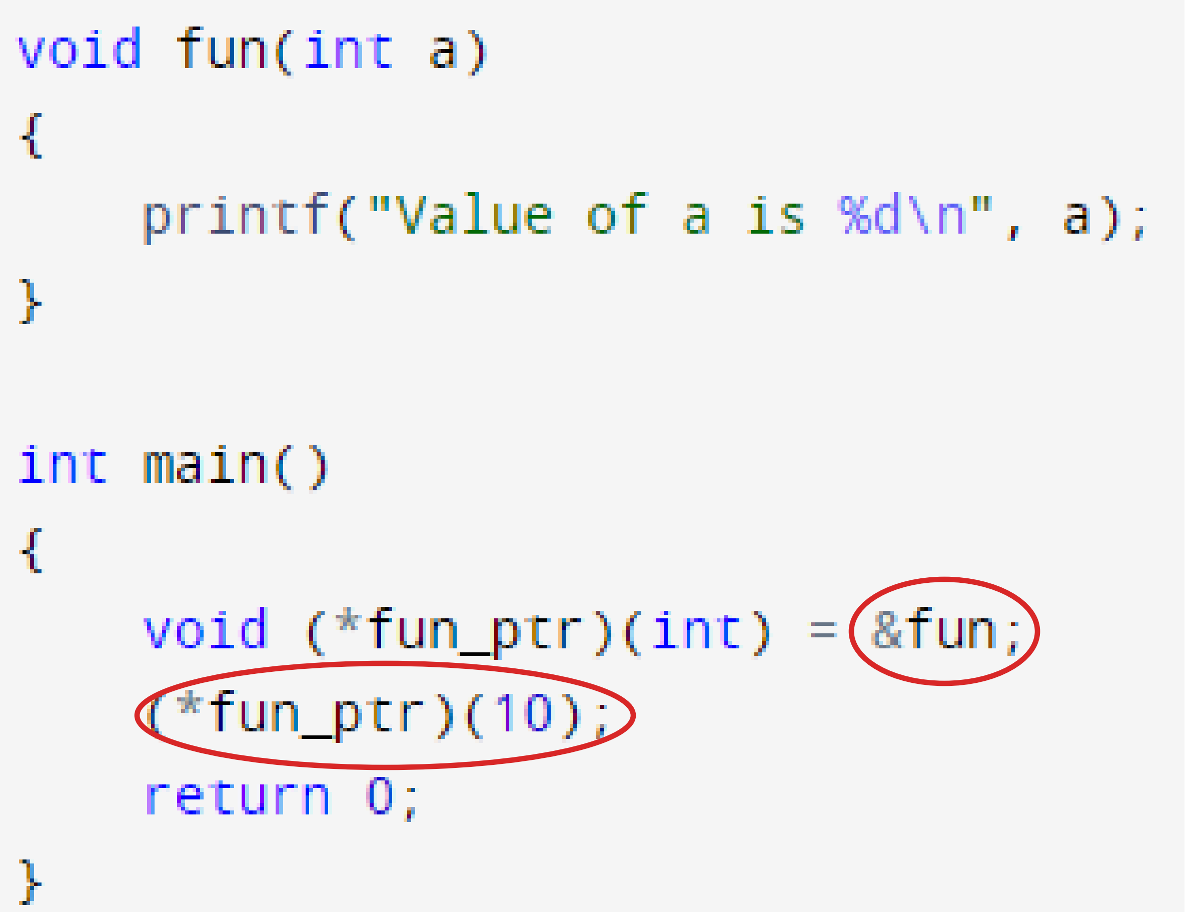


```
/tmp/NO3xBpfCRN.o
free(): invalid pointer
Aborted
```

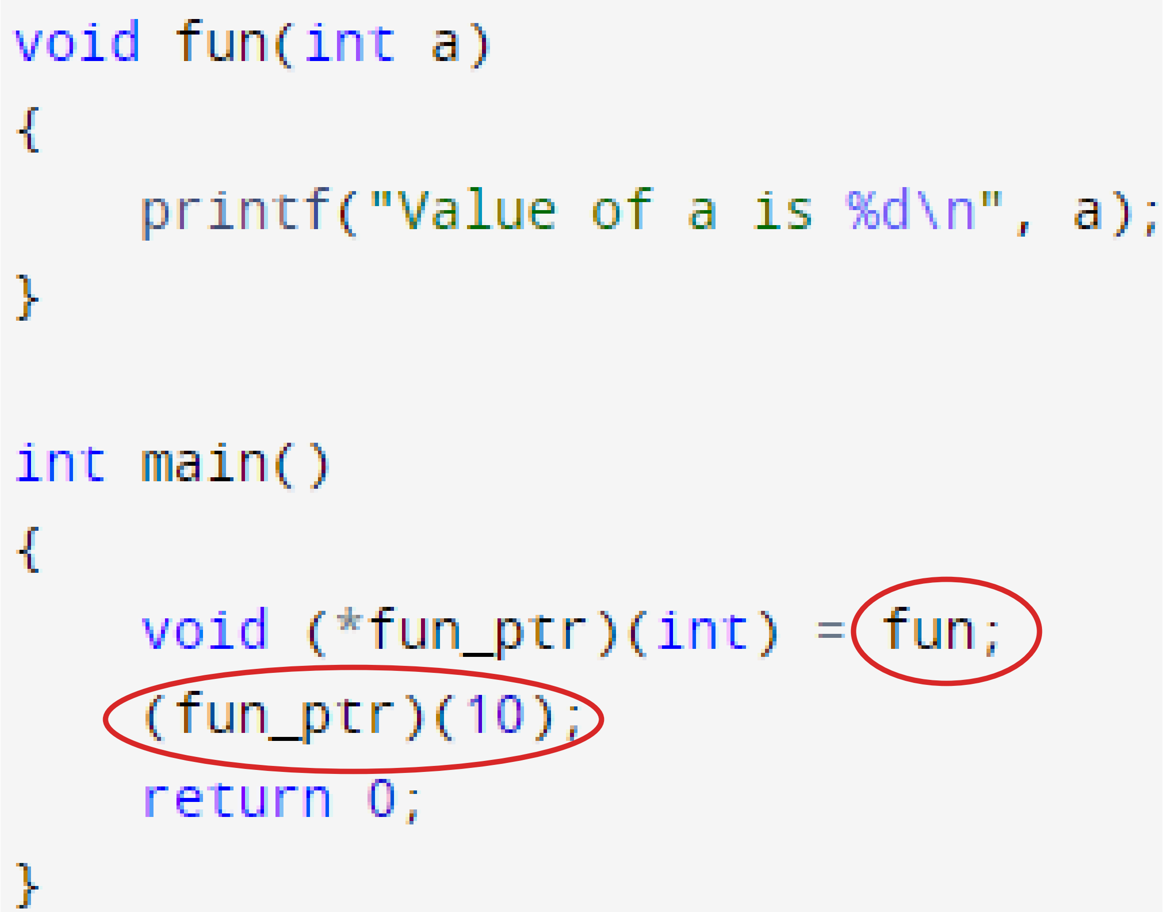
POINTER & FUNCTION: FUNCTION POINTER

Fact:

3. A function’s name can also be used to get functions’ address



```
void fun(int a)
{
    printf("Value of a is %d\n", a);
}

int main()
{
    void (*fun_ptr)(int) = &fun;
    (*fun_ptr)(10);
    return 0;
}
```



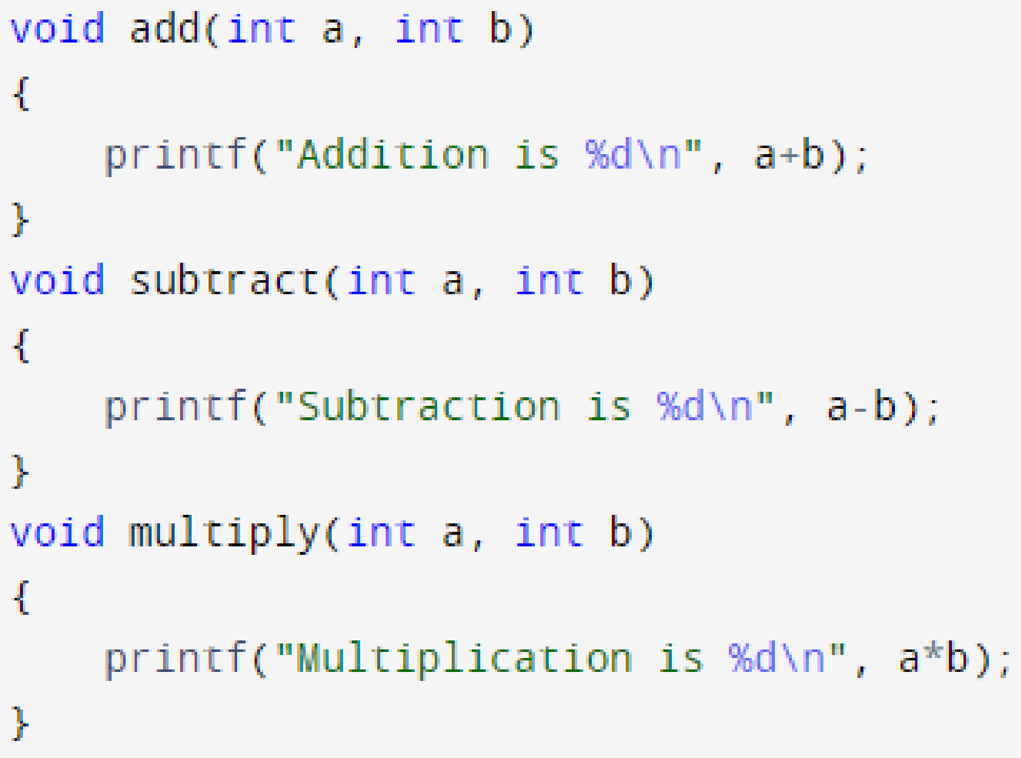
```
void fun(int a)
{
    printf("Value of a is %d\n", a);
}

int main()
{
    void (*fun_ptr)(int) = fun;
    (fun_ptr)(10);
    return 0;
}
```

POINTER & FUNCTION: FUNCTION POINTER

Fact:

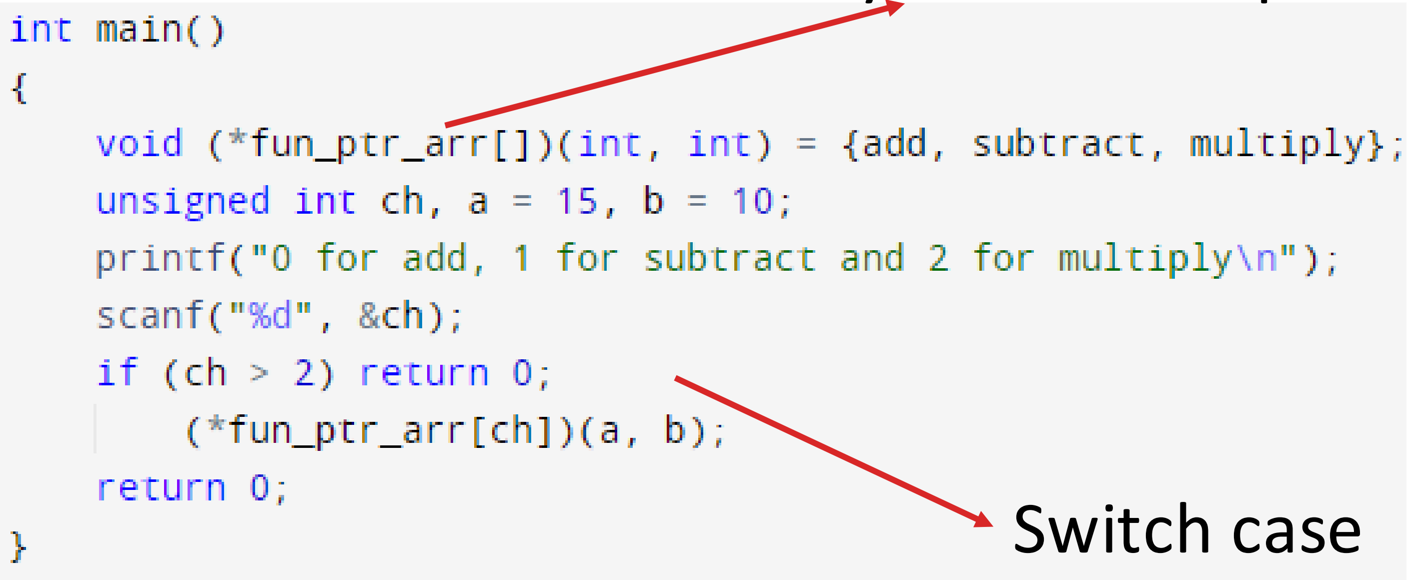
4. We can have an array of function pointers

5. Function pointer can be used in place of switch case



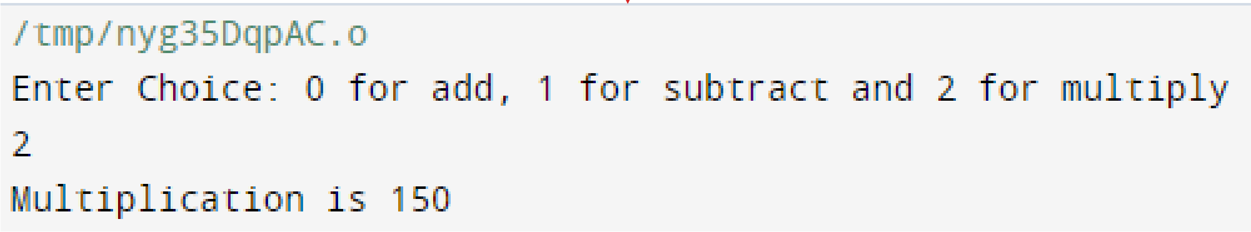
```c
void add(int a, int b)
{
    printf("Addition is %d\n", a+b);
}
void subtract(int a, int b)
{
    printf("Subtraction is %d\n", a-b);
}
void multiply(int a, int b)
{
    printf("Multiplication is %d\n", a*b);
}
```

Array of function poiter



```c
int main()
{
    void (*fun_ptr_arr[])(int, int) = {add, subtract, multiply};
    unsigned int ch, a = 15, b = 10;
    printf("0 for add, 1 for subtract and 2 for multiply\n");
    scanf("%d", &ch);
    if (ch > 2) return 0;
        (*fun_ptr_arr[ch])(a, b);
    return 0;
}
```

Switch case

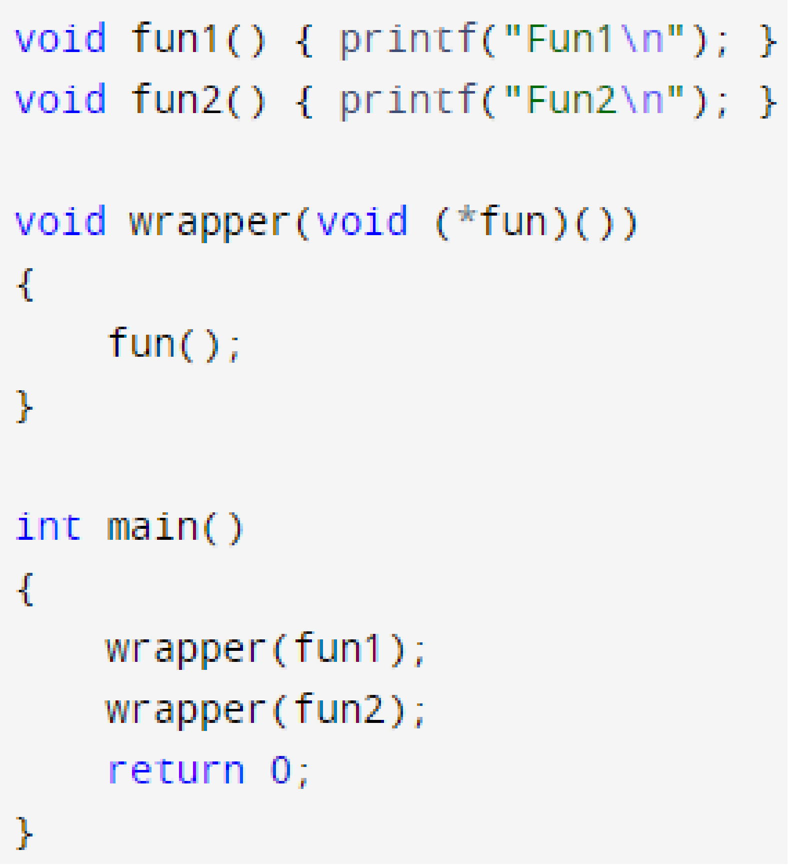


```
/tmp/nyg35DqpAC.o
Enter Choice: 0 for add, 1 for subtract and 2 for multiply
2
Multiplication is 150
```

POINTER & FUNCTION: FUNCTION POINTER

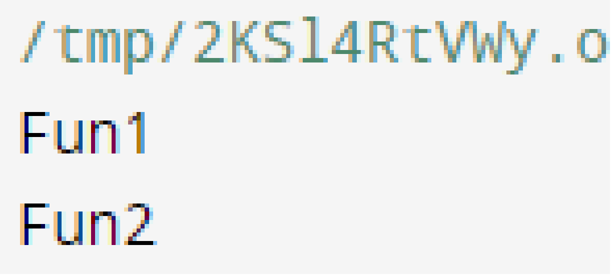
Fact:

6. Like normal data pointers, a function pointer can be passed as an argument and can also be returned from a function.



```c
void fun1() { printf("Fun1\n"); }
void fun2() { printf("Fun2\n"); }

void wrapper(void (*fun)())
{
    fun();
}

int main()
{
    wrapper(fun1);
    wrapper(fun2);
    return 0;
}
```

Application: Run any function passed (callback)

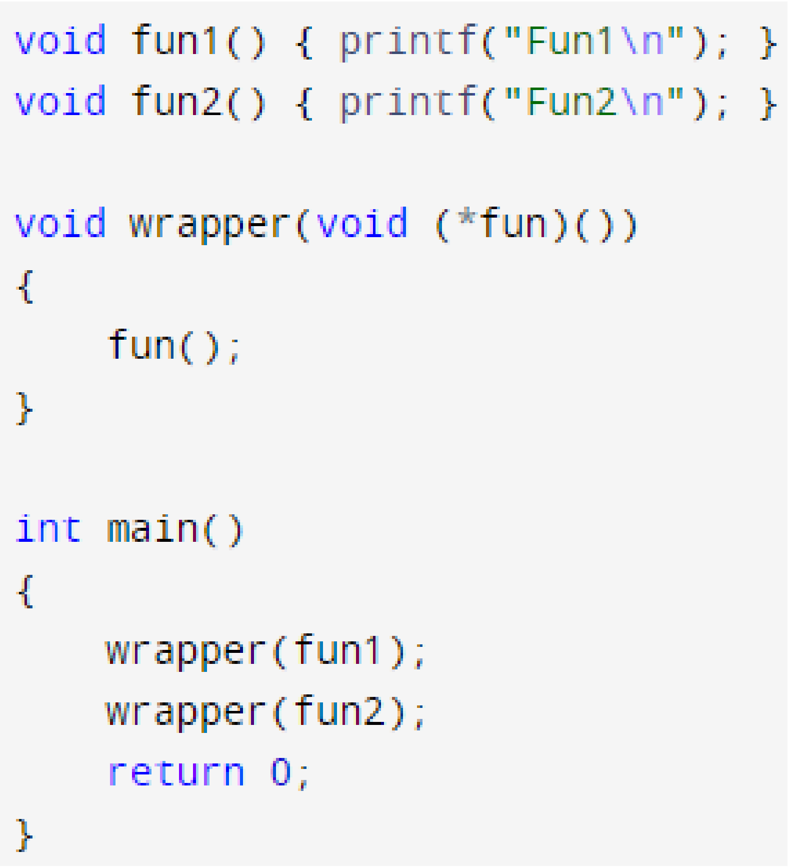


```
/tmp/2KSl4RtVWy.o
Fun1
Fun2
```

POINTER & FUNCTION: FUNCTION POINTER

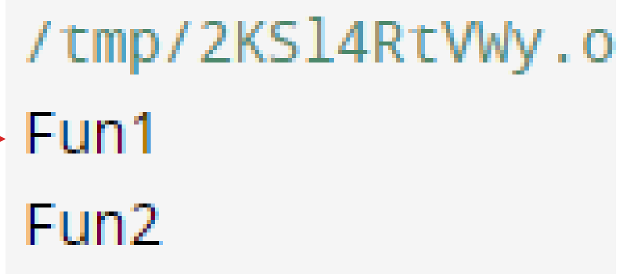
Fact:

6. Like normal data pointers, a function pointer can be passed as an argument and can also be returned from a function.



```c
void fun1() { printf("Fun1\n"); }
void fun2() { printf("Fun2\n"); }

void wrapper(void (*fun)())
{
    fun();
}

int main()
{
    wrapper(fun1);
    wrapper(fun2);
    return 0;
}
```

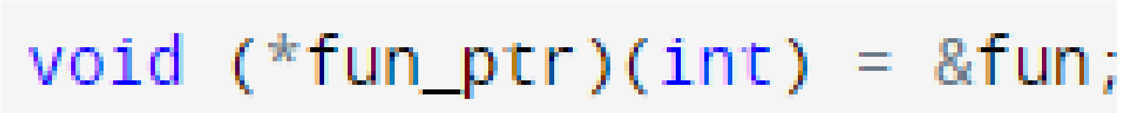
Application: Run any function passed (callback)



```
/tmp/2KSl4RtVWy.o
Fun1
Fun2
```

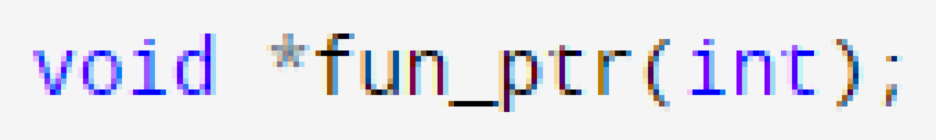
POINTER & FUNCTION: FUNCTION RETURN POINTER

Function pointer:



```
void (*fun_ptr)(int) = &fun;
```

What happen if function pointer missing the bracket???



```
void *fun_ptr(int);
```

-> It **declare function that return void pointer**

POINTER & MEMORY: DYNAMIC MEMORY ALLOCATION

Require: <stdlib.h>

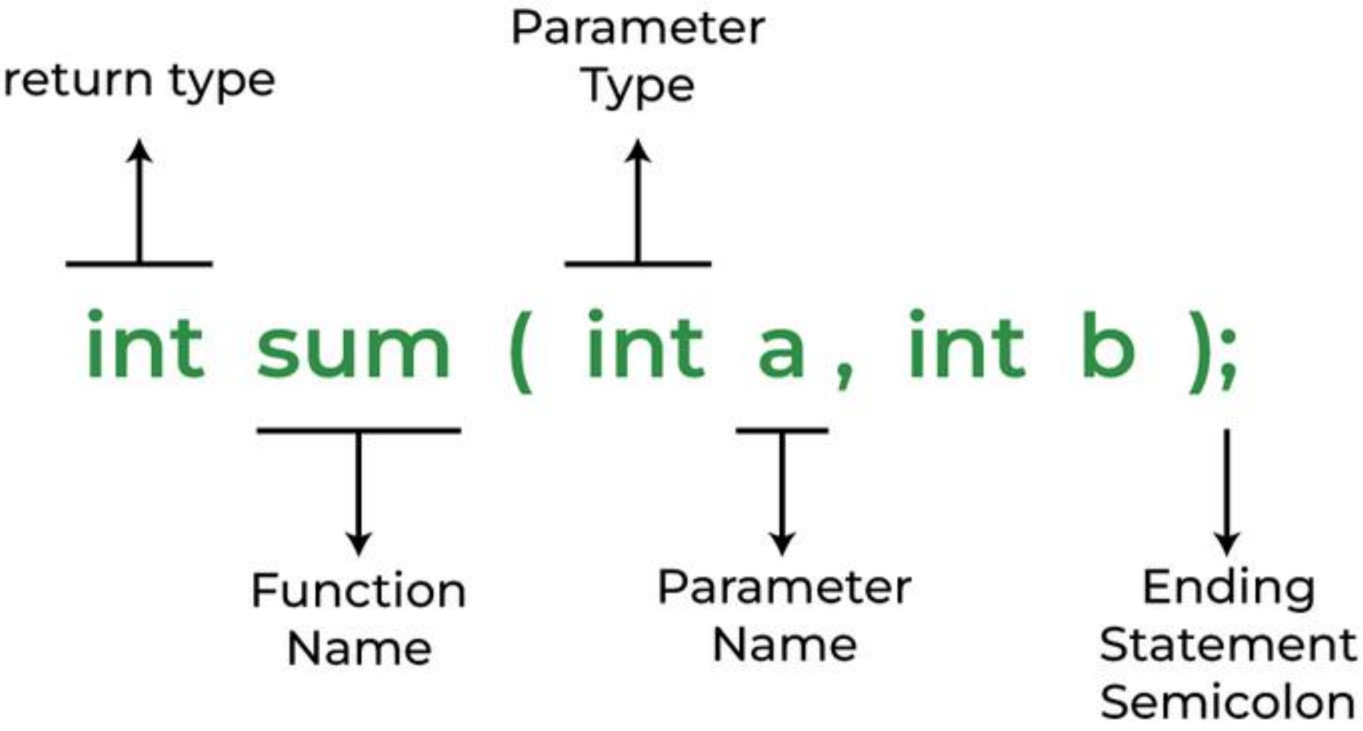
4. FUNCTION

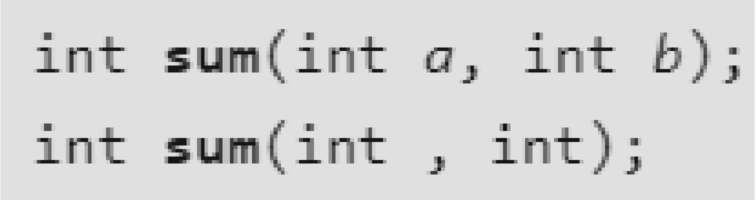
viettel

Theo cách của bạn

FUNCTION: DECLARATIONS

A function declaration tells the compiler that there is a function with the given name defined somewhere else in the program

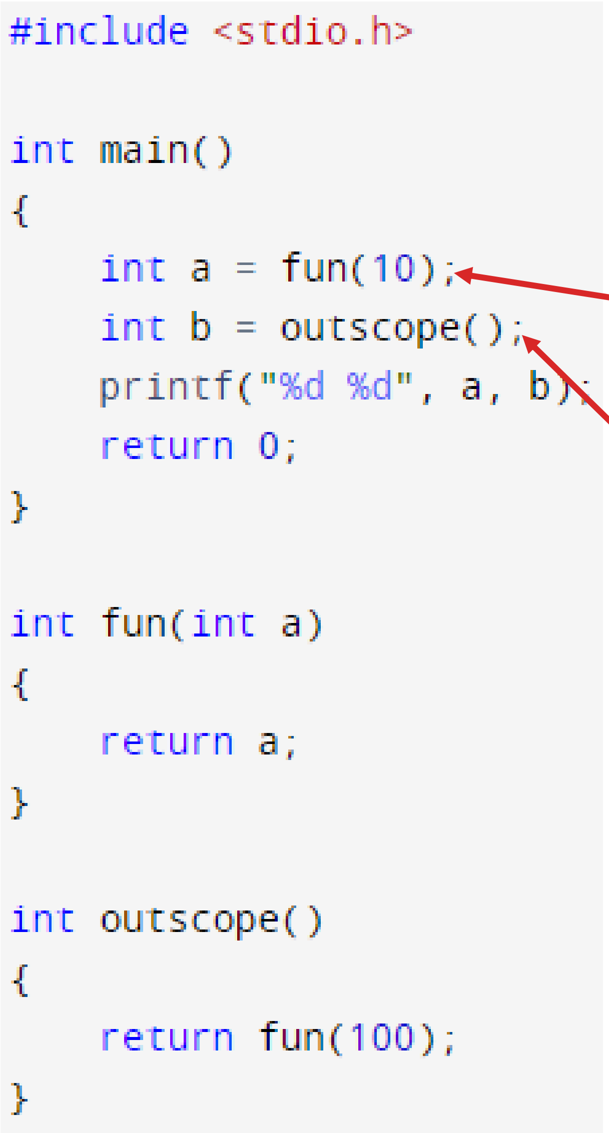




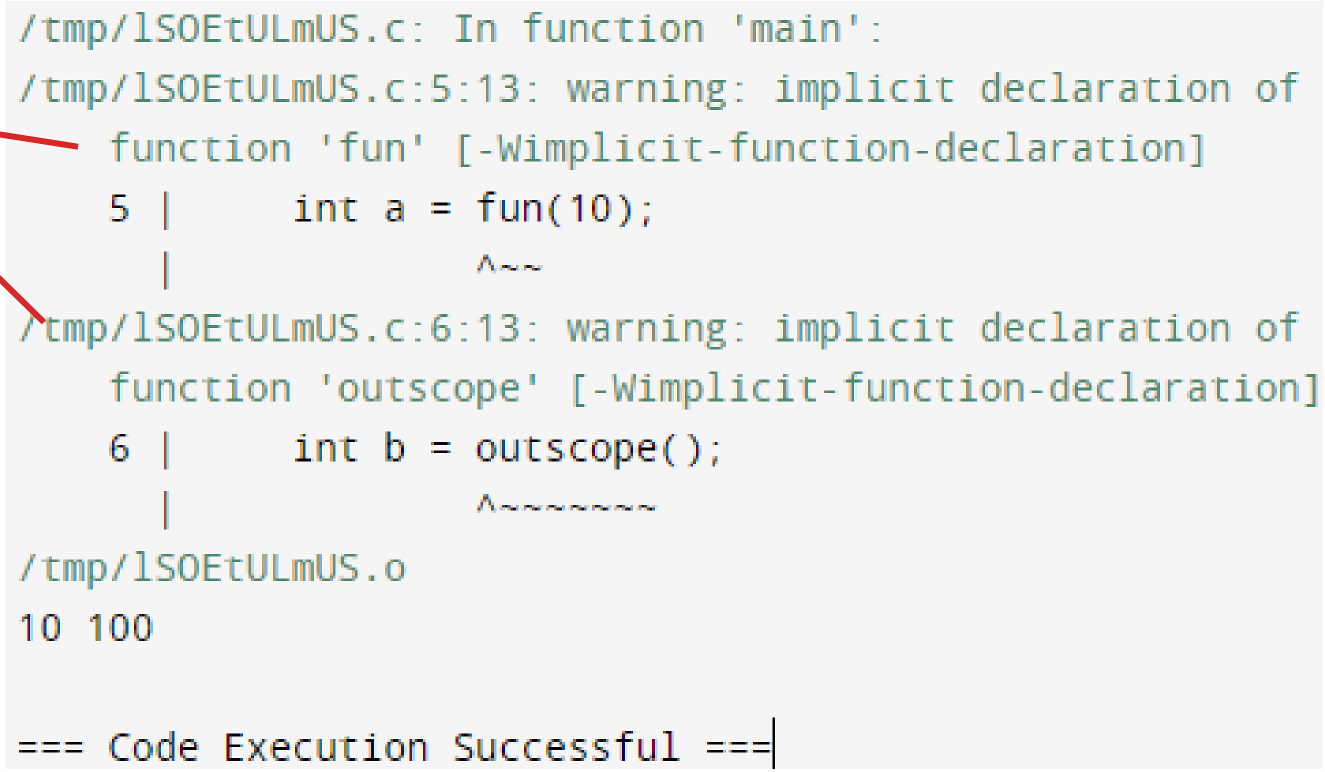
```
int sum(int a, int b);
int sum(int , int);
```

FUNCTION: DECLARATIONS

Using function without declare will raise a warning but not error



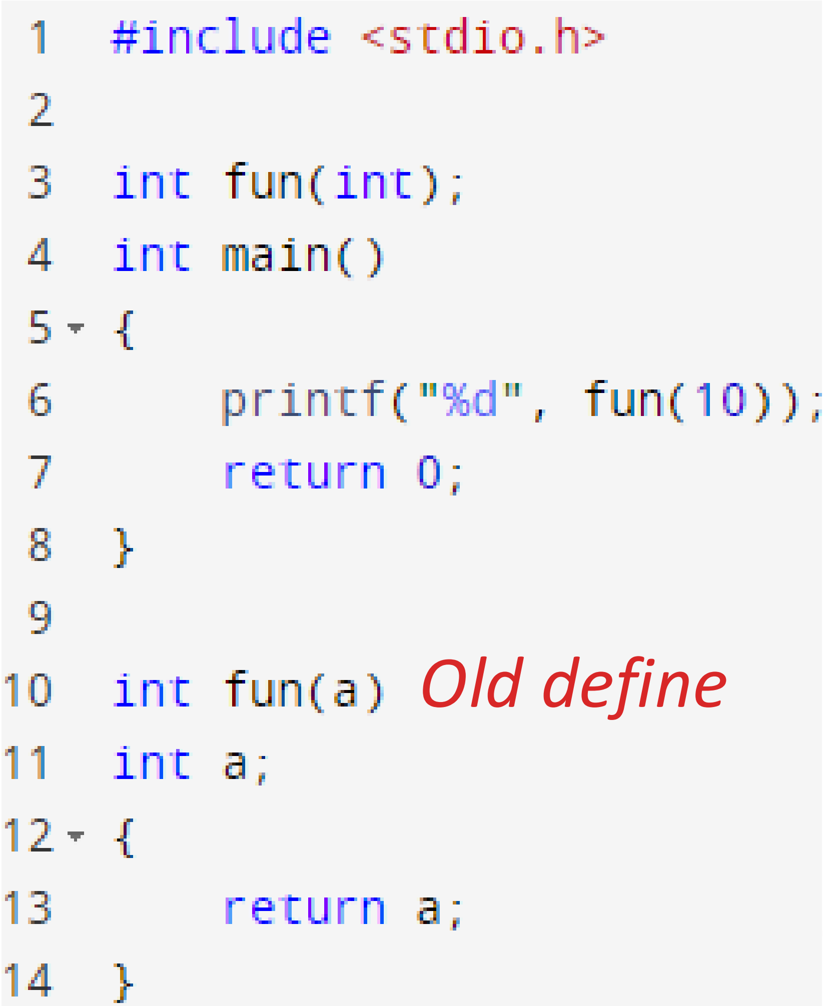
```c
#include <stdio.h>

int main()
{
    int a = fun(10);
    int b = outscope();
    printf("%d %d", a, b);
    return 0;
}

int fun(int a)
{
    return a;
}

int outscope()
{
    return fun(100);
}
```



```
/tmp/lSOEtULmUS.c: In function 'main':
/tmp/lSOEtULmUS.c:5:13: warning: implicit declaration of
    function 'fun' [-Wimplicit-function-declaration]
    5 |     int a = fun(10);
      |             ^~~
/tmp/lSOEtULmUS.c:6:13: warning: implicit declaration of
    function 'outscope' [-Wimplicit-function-declaration]
    6 |     int b = outscope();
      |             ^~~~~~~~
/tmp/lSOEtULmUS.o
10 100

=== Code Execution Successful ===
```

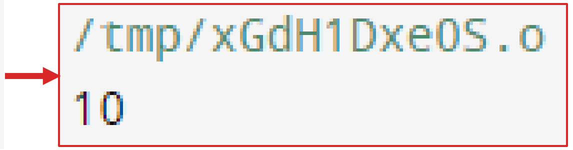
FUNCTION: DEFINITIONS

The function definition consists of actual statements which are executed when the function is called

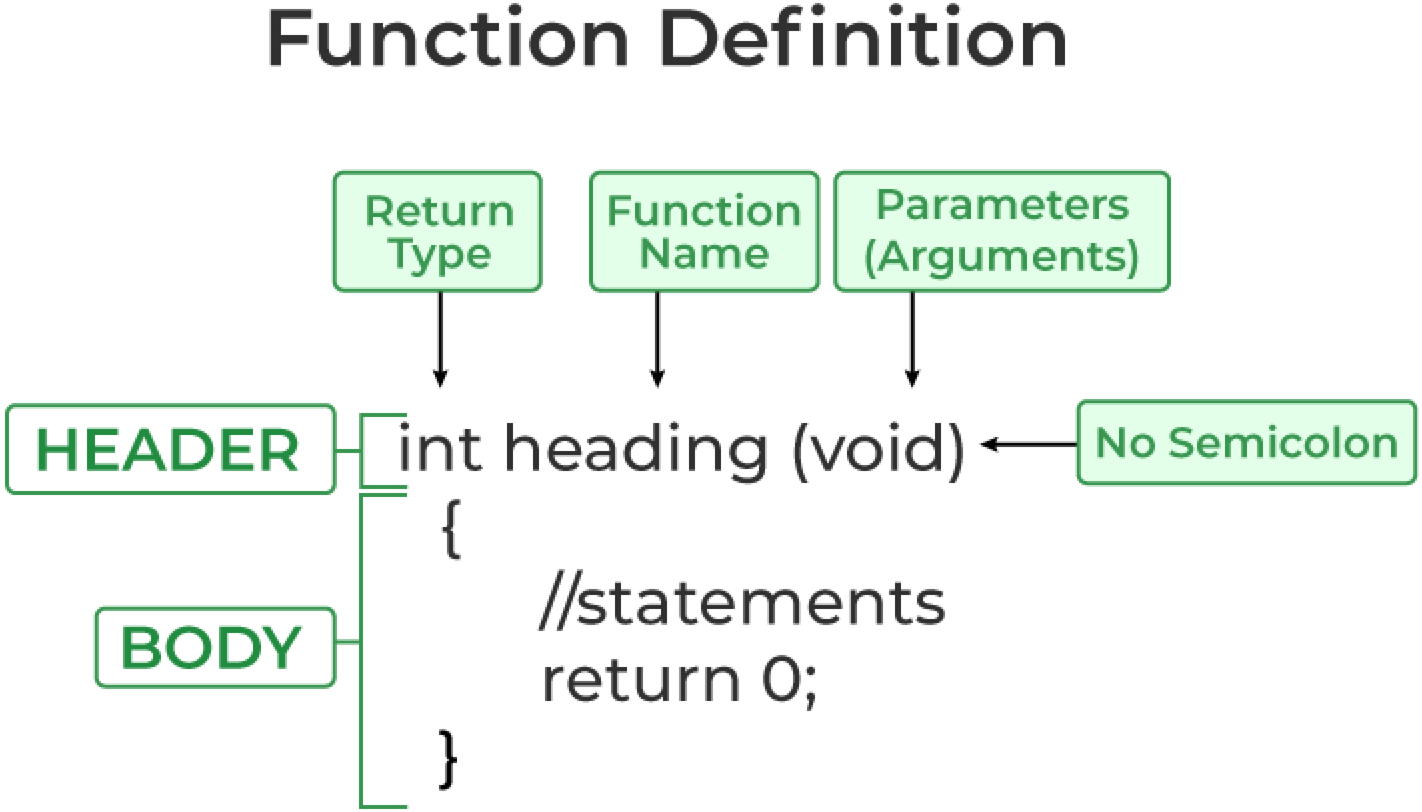


```c
#include <stdio.h>

int fun(int);
int main()
{
    printf("%d", fun(10));
    return 0;
}

int fun(a)
int a;
{
    return a;
}
```

Old define



```
/tmp/xGdH1Dxe0S.o
10
```



Function Definition

Return Type

Function Name

Parameters (Arguments)

HEADER

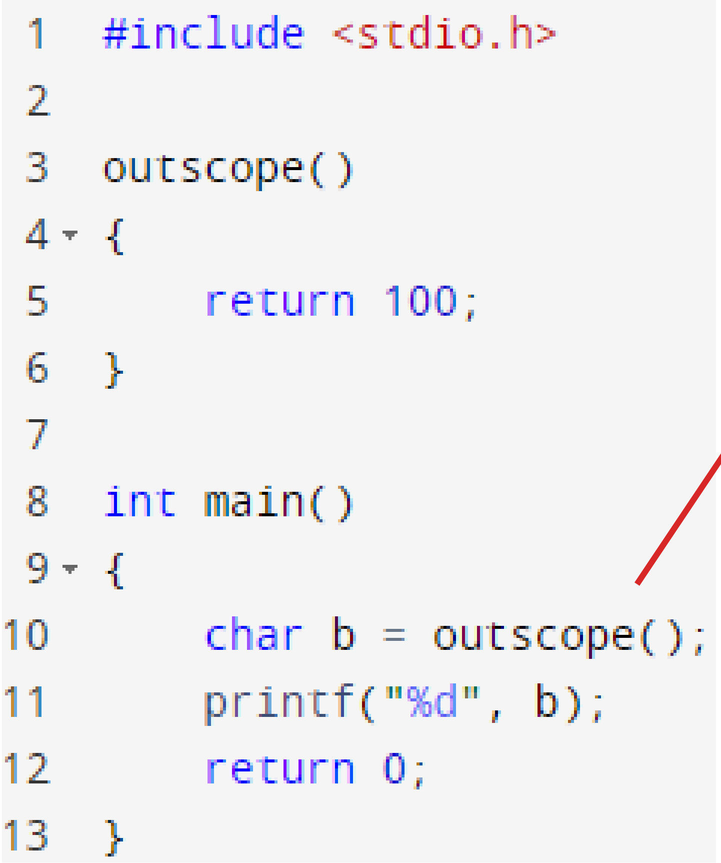
BODY

No Semicolon

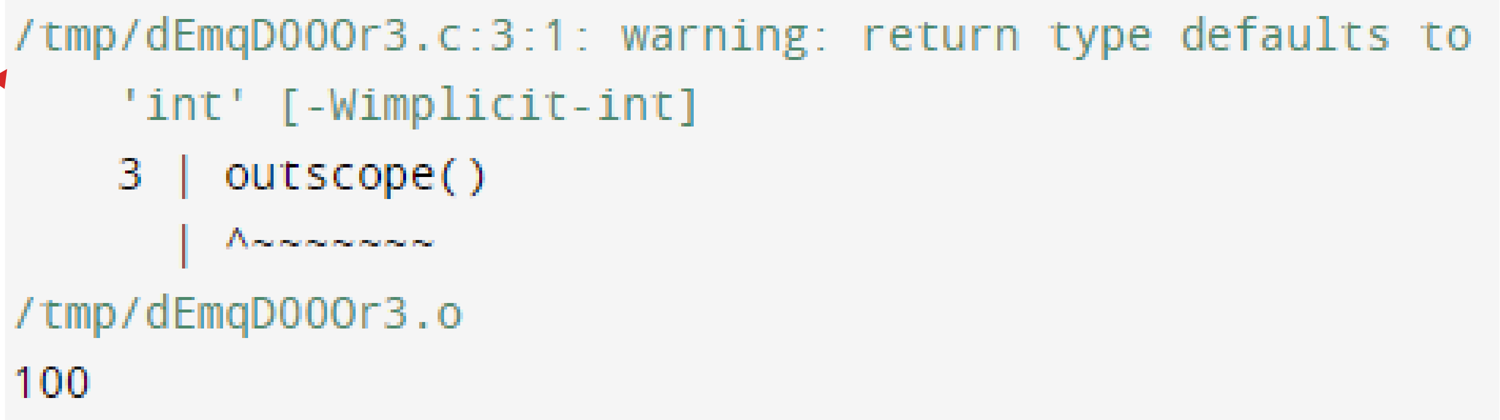
```
int heading (void)
 {
       //statements
       return 0;
 }
```

FUNCTION: IMPLICIT RETURN TYPE

In C, if we do not specify a return type, compiler assumes an implicit return type as int. However, C99 standard doesn't allow return type to be omitted even if return type is int. This was allowed in older C standard C89.



```c
#include <stdio.h>

outscope()
{
        return 100;
}

int main()
{
        char b = outscope();
        printf("%d", b);
        return 0;
}
```

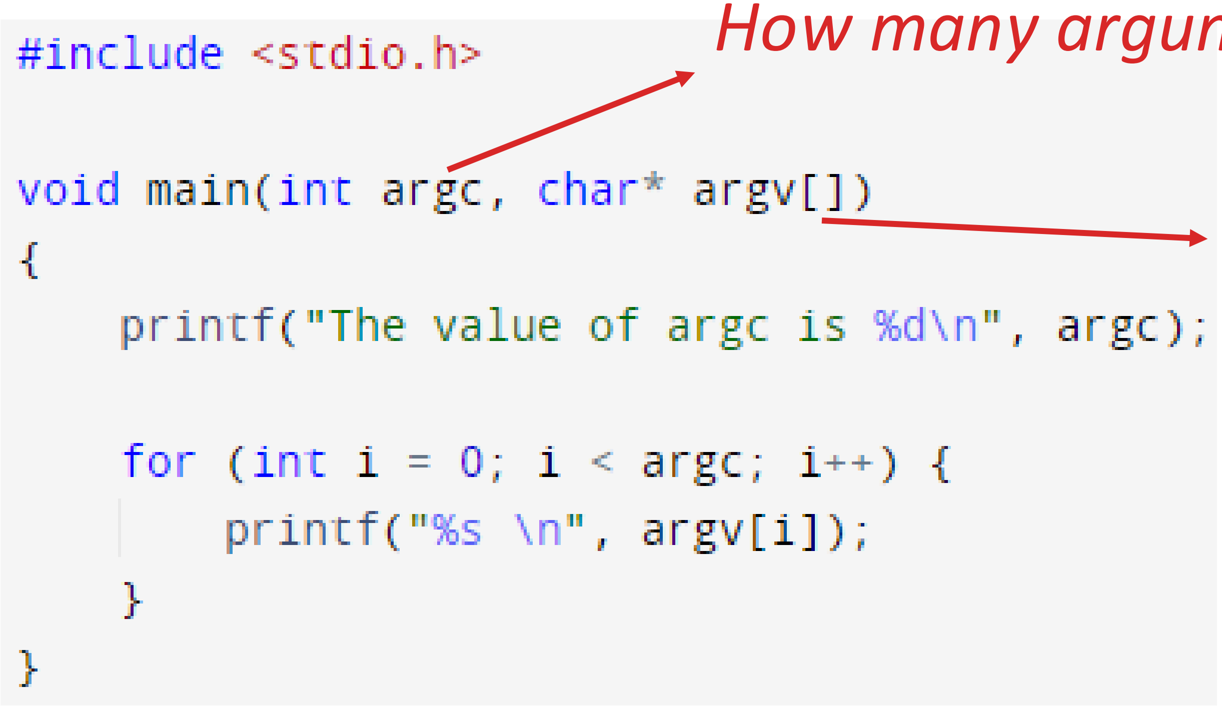


```
/tmp/dEmqD000r3.c:3:1: warning: return type defaults to
    'int' [-Wimplicit-int]
    3 | outscope()
      | ^~~~~~~~~
/tmp/dEmqD000r3.o
100
```

FUNCTION: MAIN FUNCTION

main() function called first in program.

main() has it own parameters (commad line arguments) and return type as other function:

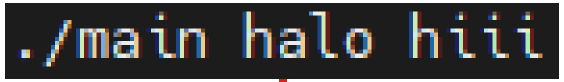


```c
#include <stdio.h>

void main(int argc, char* argv[])
{
    printf("The value of argc is %d\n", argc);

    for (int i = 0; i < argc; i++) {
        printf("%s \n", argv[i]);
    }
}
```

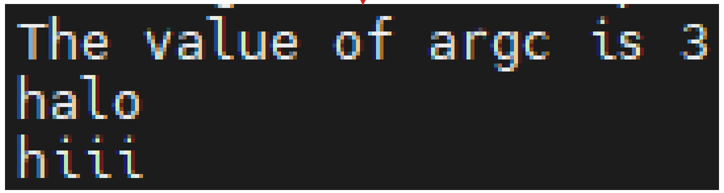
How many argument passed + 1

String array argument passed

Return type (void is no return)



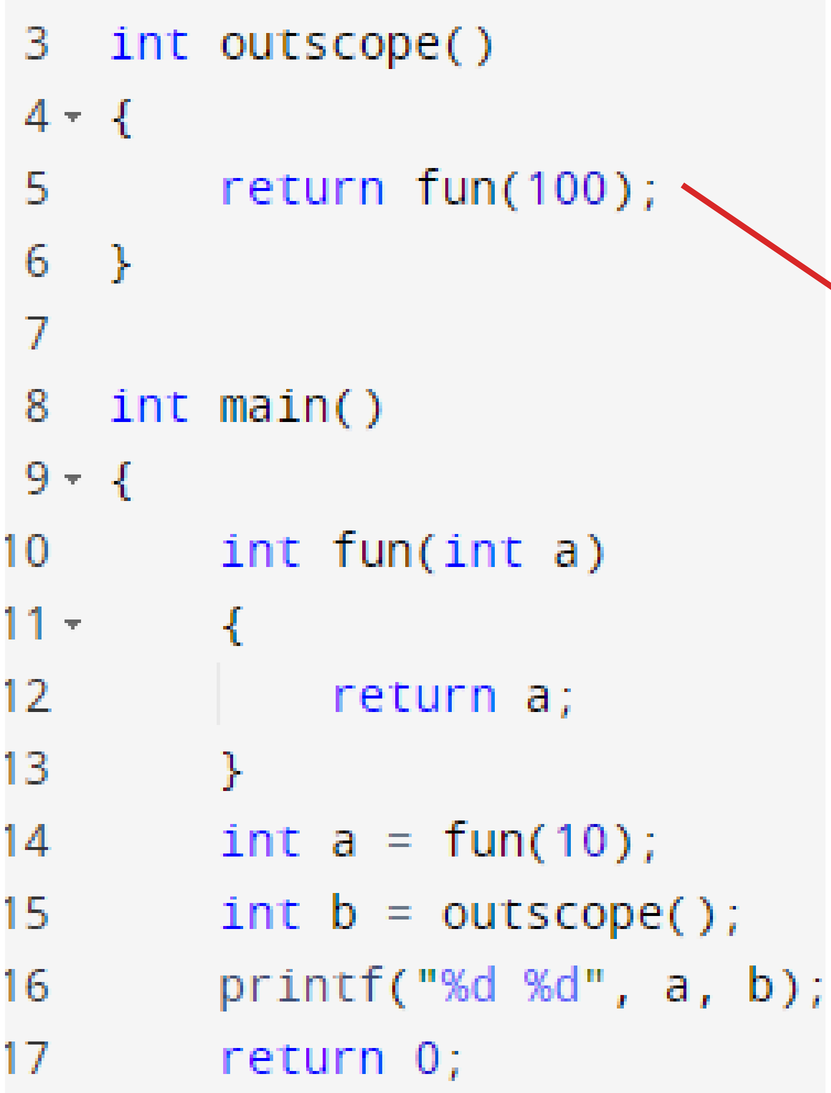
```
./main halo hiii
```



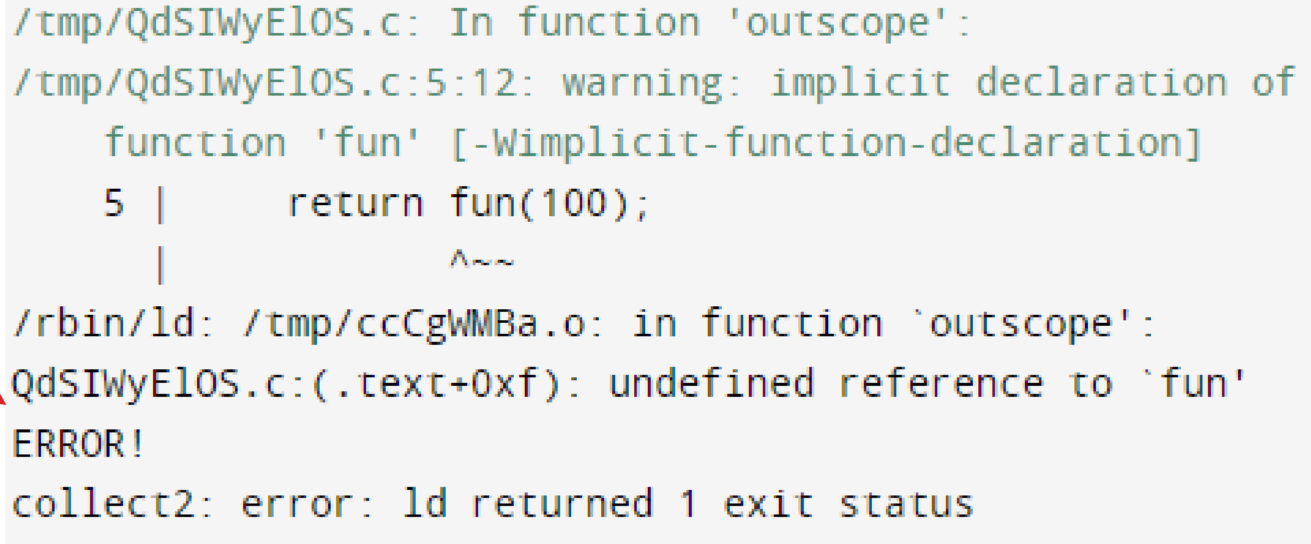
```
The value of argc is 3
halo
hiii
```

FUNCTION: NESTED FUNCTION

The function definition in other function will private in that function



```c
int outscope()
{
    return fun(100);
}

int main()
{
    int fun(int a)
    {
        return a;
    }
    int a = fun(10);
    int b = outscope();
    printf("%d %d", a, b);
    return 0;
```

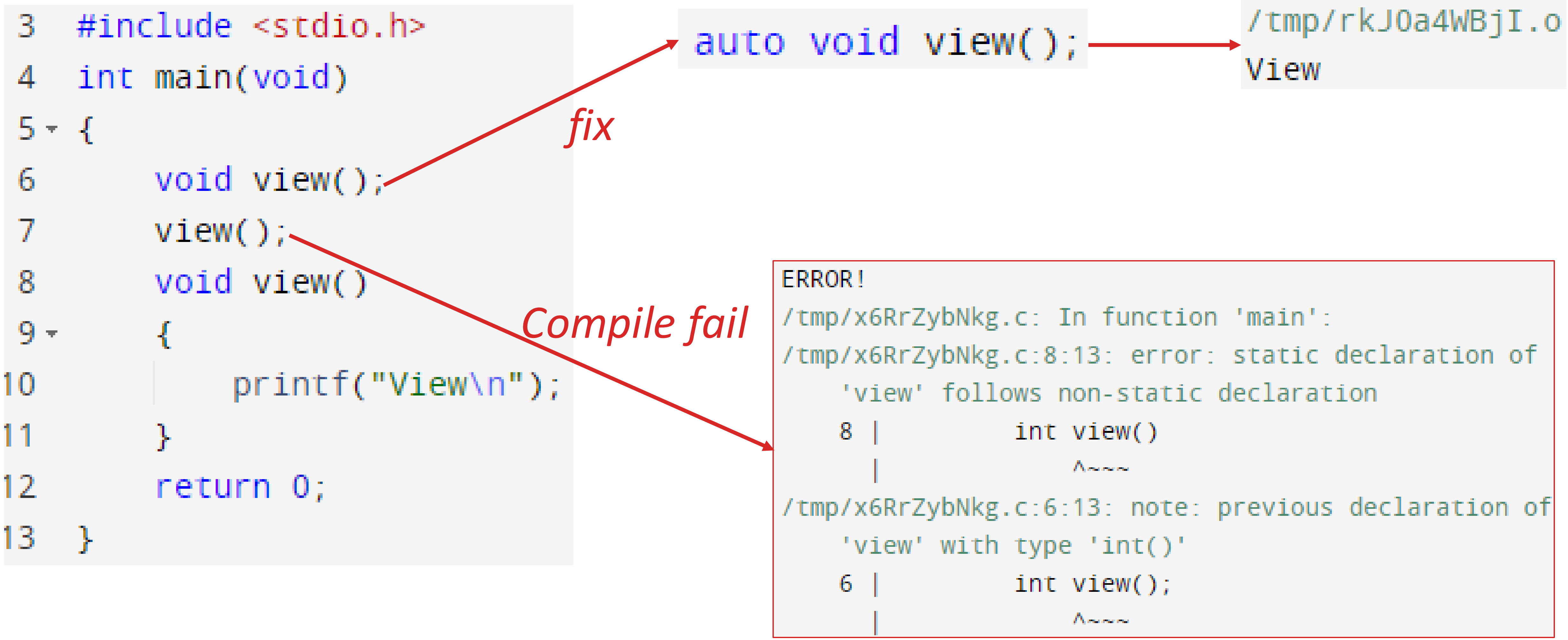


```
/tmp/QdSIWyElOS.c: In function 'outscope':
/tmp/QdSIWyElOS.c:5:12: warning: implicit declaration of
    function 'fun' [-Wimplicit-function-declaration]
    5 |     return fun(100);
      |            ^~~
/rbin/ld: /tmp/ccCgWMBa.o: in function `outscope':
QdSIWyElOS.c:(.text+0xf): undefined reference to `fun'
ERROR!
collect2: error: ld returned 1 exit status
```

Can compile but can't run

FUNCTION: NESTED FUNCTION

Other error: Declare and call function in same block, compiler understand it is 2 declare but not call



```
3   #include <stdio.h>
4   int main(void)
5   {
6       void view();
7       view();
8       void view()
9       {
10          printf("View\n");
11      }
12      return 0;
13  }
```

fix

```
auto void view();
```

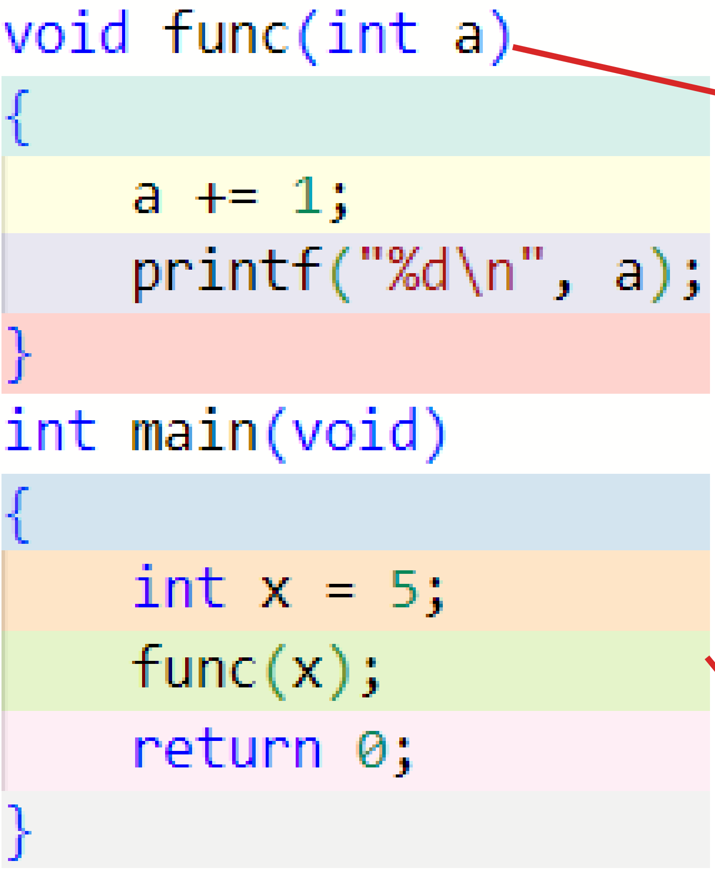
```
/tmp/rkJ0a4WBjI.o
View
```

Compile fail

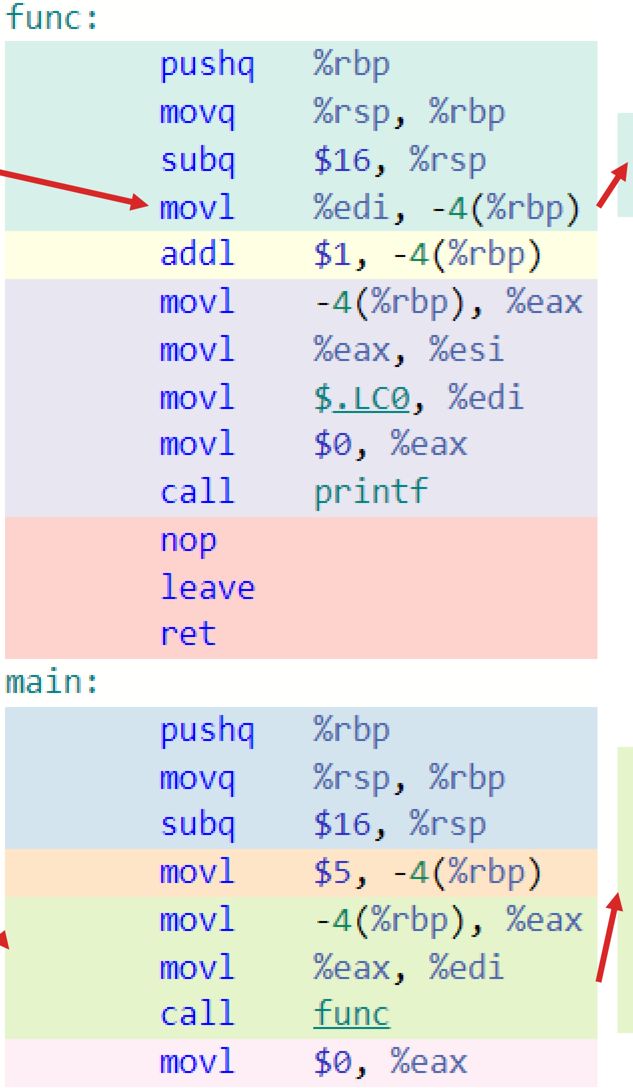
```
ERROR!
/tmp/x6RrZybNkg.c: In function 'main':
/tmp/x6RrZybNkg.c:8:13: error: static declaration of
    'view' follows non-static declaration
    8 |         int view()
      |             ^~~~
/tmp/x6RrZybNkg.c:6:13: note: previous declaration of
    'view' with type 'int()'
    6 |         int view();
      |             ^~~~
```

FUNCTION: PARAMETER PASSING – PASS BY VALUE

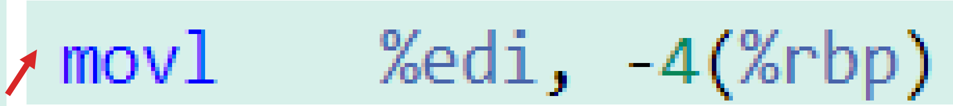
Changes made to formal parameters do not get transmitted back to the caller.



```c
void func(int a)
{
    a += 1;
    printf("%d\n", a);
}
int main(void)
{
    int x = 5;
    func(x);
    return 0;
}
```

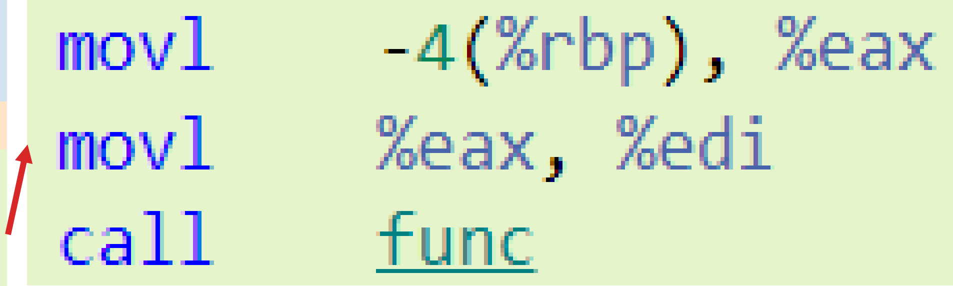


```
func:
        pushq   %rbp
        movq    %rsp, %rbp
        subq    $16, %rsp
        movl    %edi, -4(%rbp)
        addl    $1, -4(%rbp)
        movl    -4(%rbp), %eax
        movl    %eax, %esi
        movl    $.LC0, %edi
        movl    $0, %eax
        call    printf
        nop
        leave
        ret
main:
        pushq   %rbp
        movq    %rsp, %rbp
        subq    $16, %rsp
        movl    $5, -4(%rbp)
        movl    -4(%rbp), %eax
        movl    %eax, %edi
        call    func
        movl    $0, %eax
```



```
movl    %edi, -4(%rbp)
```

Get value a = %edi

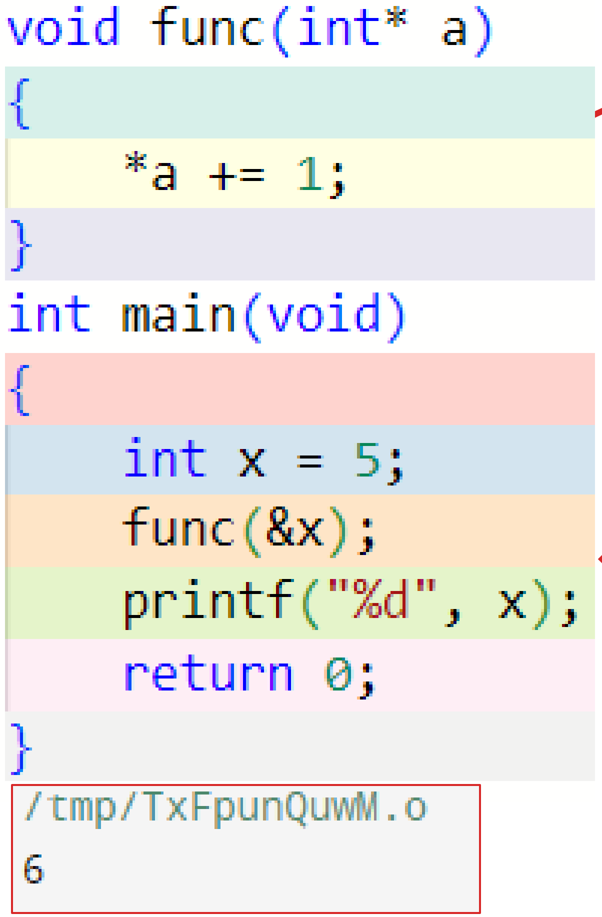


```
movl    -4(%rbp), %eax
movl    %eax, %edi
call    func
```

%edi = value x

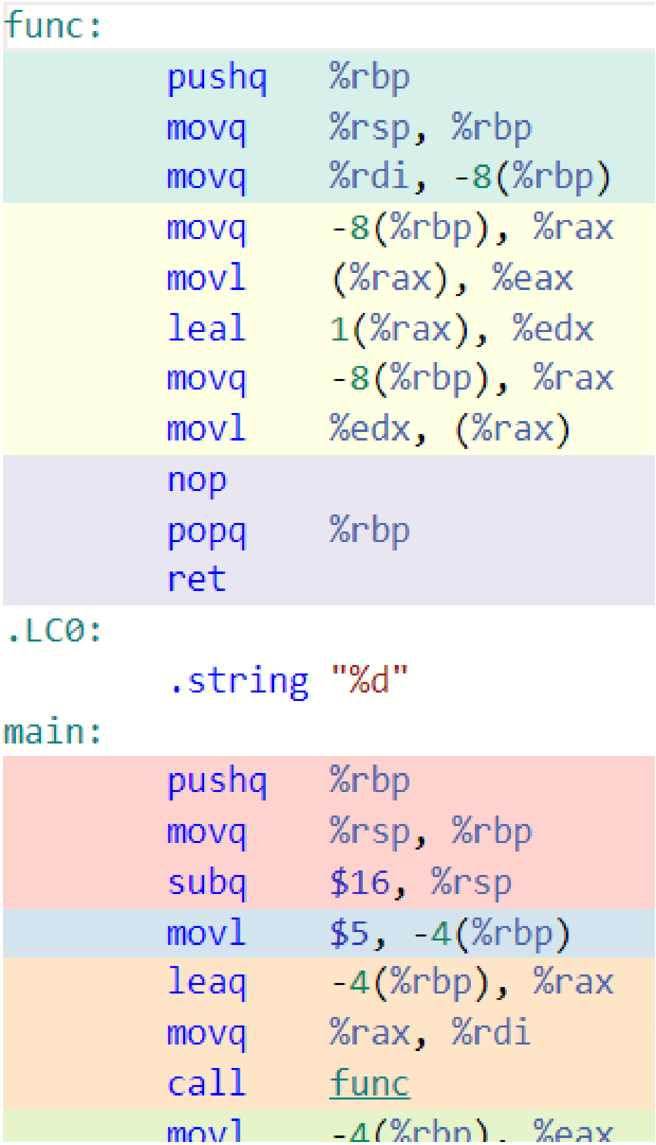
FUNCTION: PARAMETER PASSING – PASS BY REFERENCE

Pass memory address (pointer) of a variable **allows the function to access and modify the content** at that particular memory location.



```
void func(int* a)
{
    *a += 1;
}
int main(void)
{
    int x = 5;
    func(&x);
    printf("%d", x);
    return 0;
}
```

```
/tmp/TxFpunQuwM.o
6
```



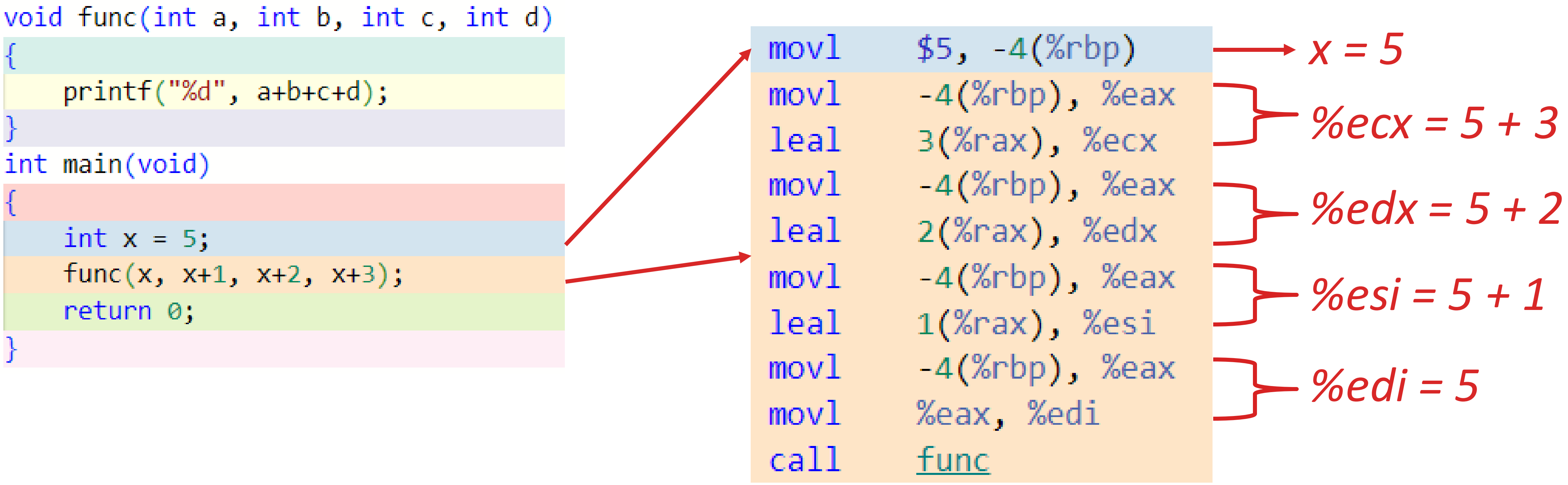
```
func:
        pushq   %rbp
        movq    %rsp, %rbp
        movq    %rdi, -8(%rbp)
        movq    -8(%rbp), %rax
        movl    (%rax), %eax
        leal    1(%rax), %edx
        movq    -8(%rbp), %rax
        movl    %edx, (%rax)
        nop
        popq    %rbp
        ret
.LC0:
        .string "%d"
main:
        pushq   %rbp
        movq    %rsp, %rbp
        subq    $16, %rsp
        movl    $5, -4(%rbp)
        leaq    -4(%rbp), %rax
        movq    %rax, %rdi
        call    func
        movl    -4(%rbp), %eax
```

a = %rdi = address of x

%rdi = address of x

FUNCTION: PARAMETER PASSING – ORDERING

Function call parameter from **right to left:**

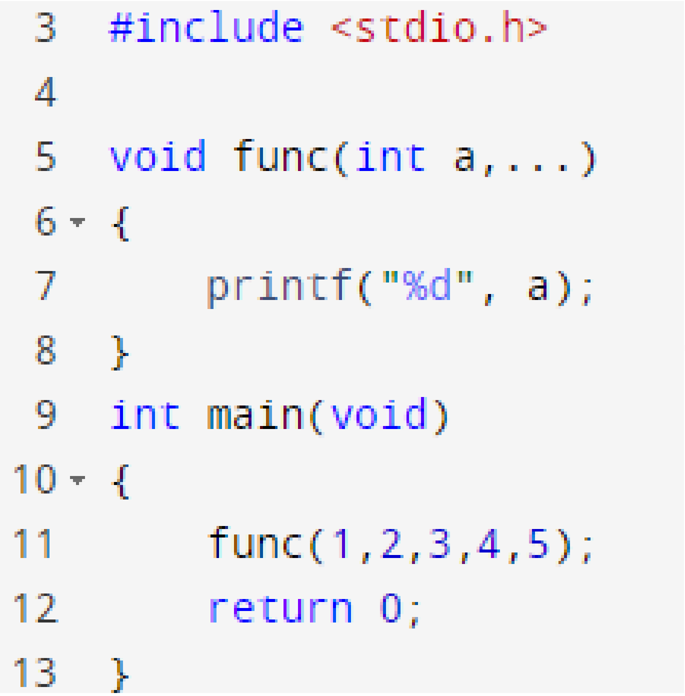


```
void func(int a, int b, int c, int d)
{
    printf("%d", a+b+c+d);
}
int main(void)
{
    int x = 5;
    func(x, x+1, x+2, x+3);
    return 0;
}
```

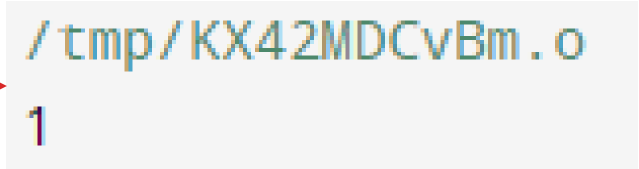
```
movl    $5, -4(%rbp)
movl    -4(%rbp), %eax
leal    3(%rax), %ecx
movl    -4(%rbp), %eax
leal    2(%rax), %edx
movl    -4(%rbp), %eax
leal    1(%rax), %esi
movl    -4(%rbp), %eax
movl    %eax, %edi
call    func
```

FUNCTION: ELLIPSIS ARGUMENT

An **ellipsis is** used to represent a **variable number of parameters** to a function, must always be the last argument.



```c
#include <stdio.h>

void func(int a,...)
{
    printf("%d", a);
}
int main(void)
{
    func(1,2,3,4,5);
    return 0;
}
```

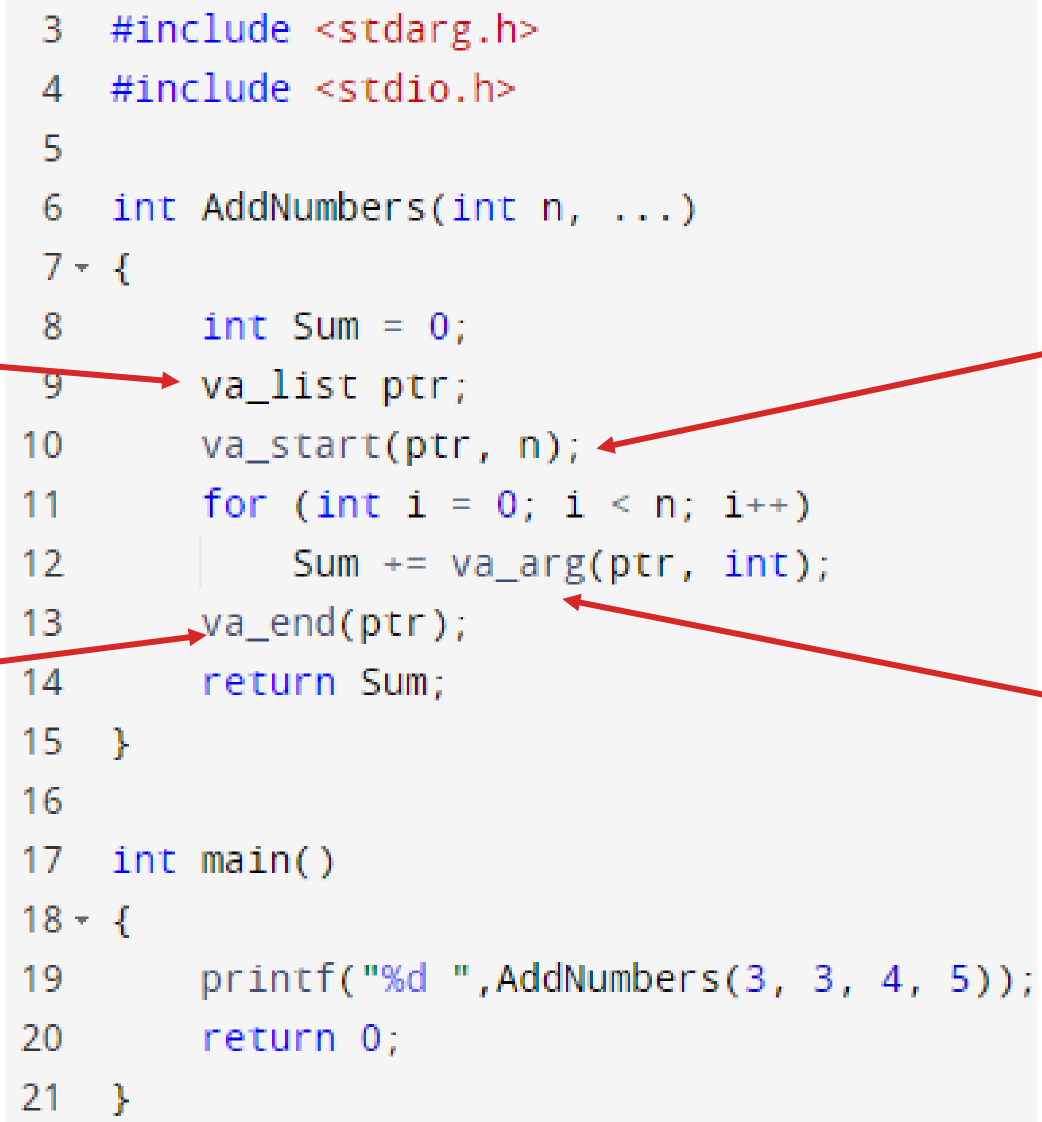


```
/tmp/KX42MDCvBm.o
1
```

FUNCTION: VARIADIC FUNCTION

Variadic functions are functions that can take a variable number of arguments.

Require: **<stdarg.h>**



```c
#include <stdarg.h>
#include <stdio.h>

int AddNumbers(int n, ...)
{
    int Sum = 0;
    va_list ptr;
    va_start(ptr, n);
    for (int i = 0; i < n; i++)
        Sum += va_arg(ptr, int);
    va_end(ptr);
    return Sum;
}

int main()
{
    printf("%d ",AddNumbers(3, 3, 4, 5));
    return 0;
}
```

Declaring pointer to the argument list

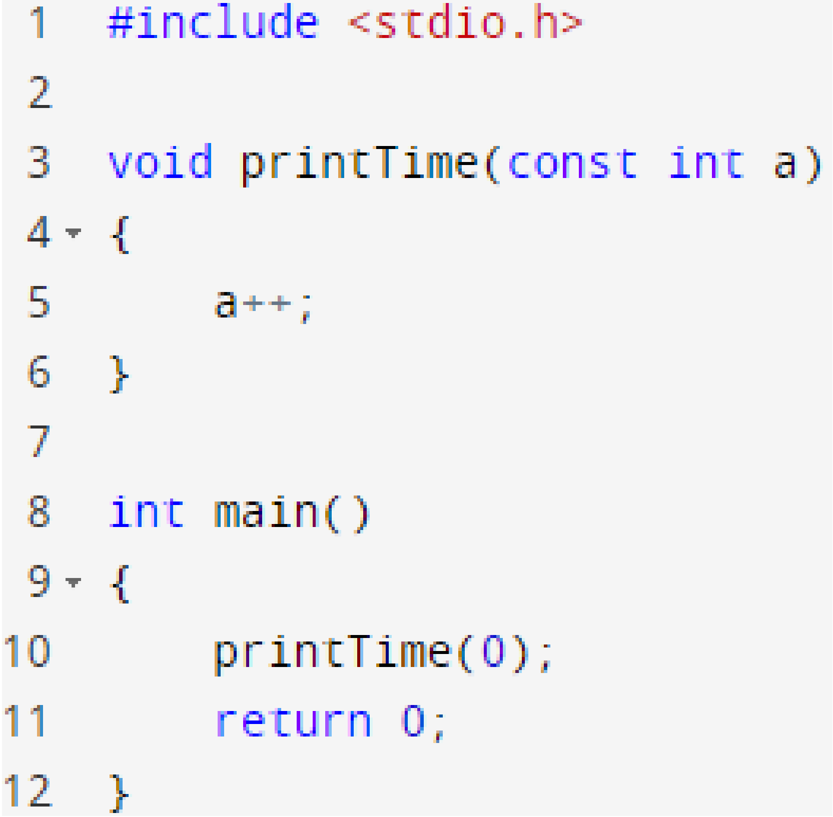
Initializing argument to the list pointer

This ends the traversal of the variadic function arguments

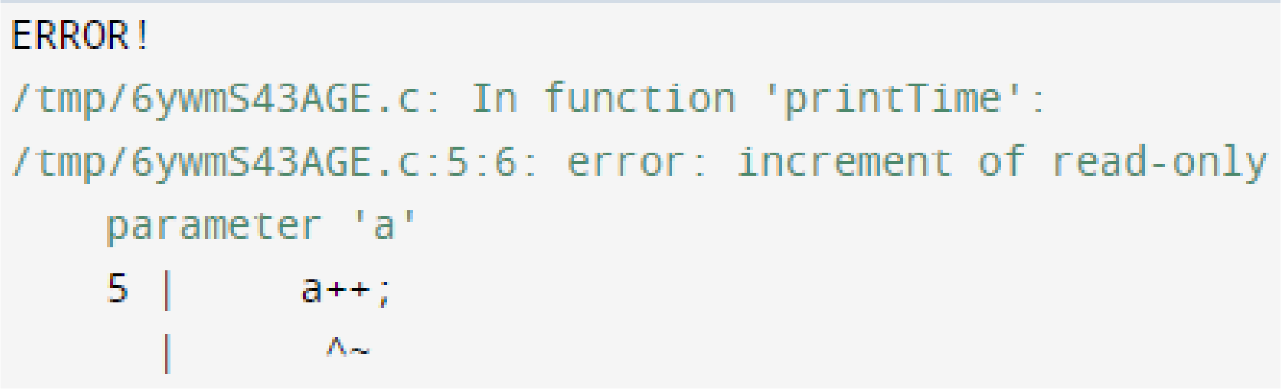
Accessing current variable and pointing to next one

FUNCTION: CONST PARAMETER

Qualify a function parameter using the const keyword indicates that the function will treat the argument that is passed as a constant.



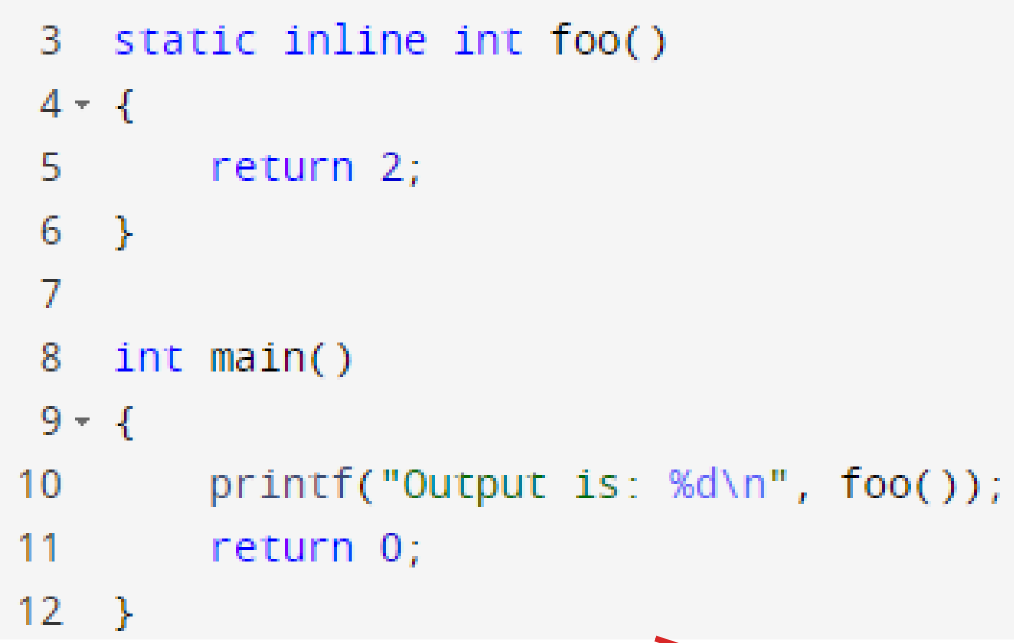
```
1   #include <stdio.h>
2
3   void printTime(const int a)
4   {
5       a++;
6   }
7
8   int main()
9   {
10      printTime(0);
11      return 0;
12  }
```



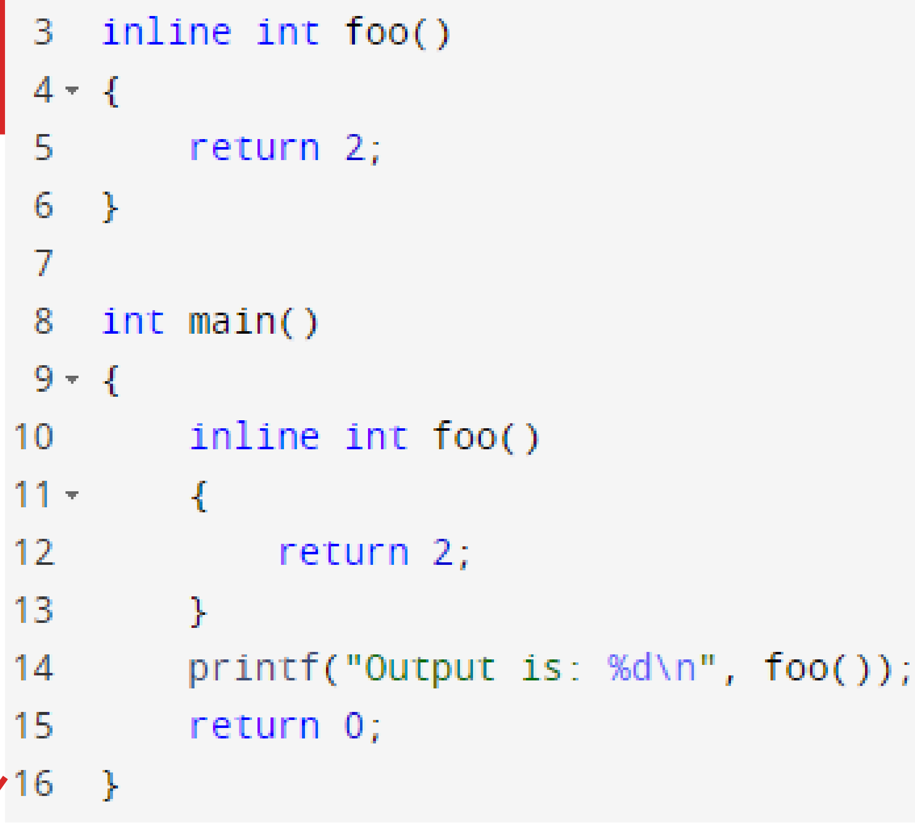
```
ERROR!
/tmp/6ywmS43AGE.c: In function 'printTime':
/tmp/6ywmS43AGE.c:5:6: error: increment of read-only
    parameter 'a'
    5 |     a++;
      |      ^~
```

FUNCTION: INLINE FUNCTION

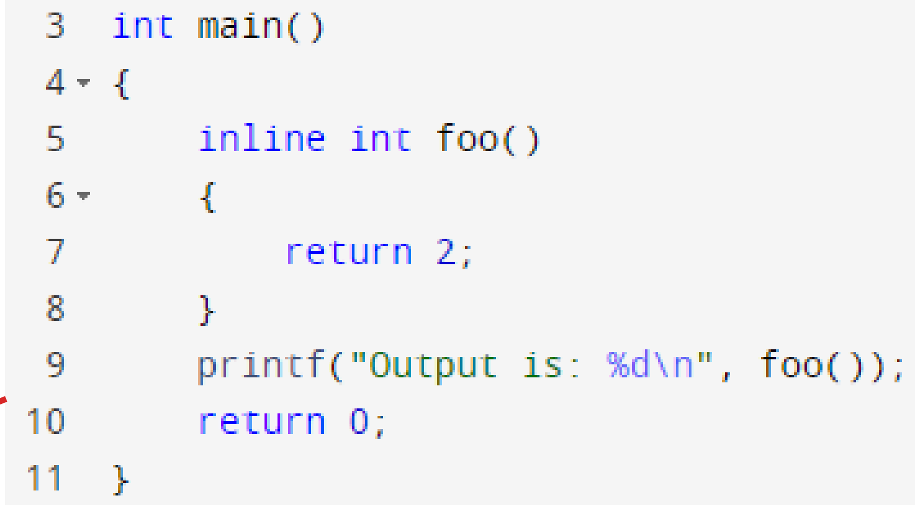
By declaring a function inline, you can direct GCC to make calls to that function faster. One way GCC can achieve this is to integrate that function's code into the code for its callers.



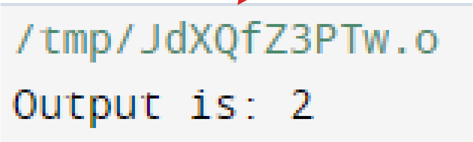
```c
static inline int foo()
{
    return 2;
}

int main()
{
    printf("Output is: %d\n", foo());
    return 0;
}
```



```c
inline int foo()
{
    return 2;
}

int main()
{
    inline int foo()
    {
        return 2;
    }
    printf("Output is: %d\n", foo());
    return 0;
}
```



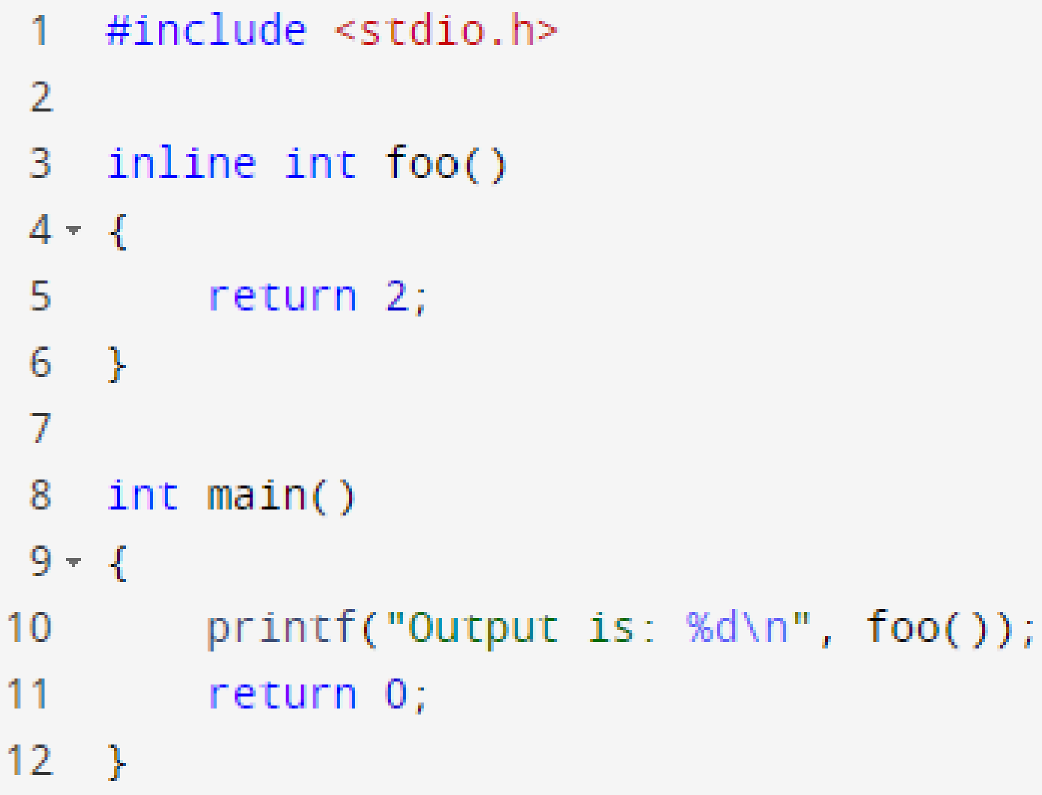
```c
int main()
{
    inline int foo()
    {
        return 2;
    }
    printf("Output is: %d\n", foo());
    return 0;
}
```



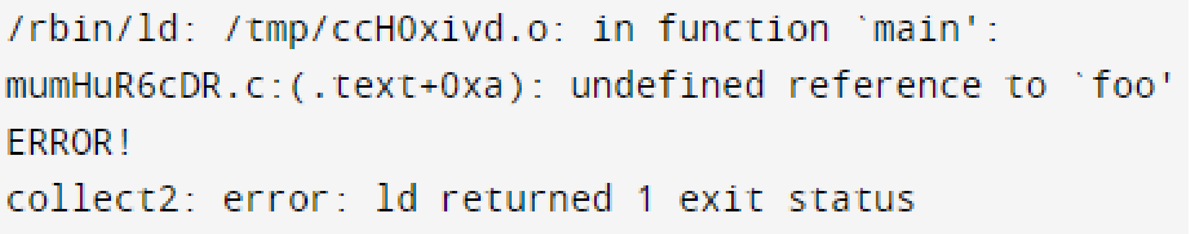
```
/tmp/JdXQfZ3PTw.o
Output is: 2
```

FUNCTION: INLINE FUNCTION

Inline Function are those function whose definitions are substituted at the place where its function call is happened



```c
#include <stdio.h>

inline int foo()
{
    return 2;
}

int main()
{
    printf("Output is: %d\n", foo());
    return 0;
}
```



```
/rbin/ld: /tmp/ccH0xivd.o: in function `main':
mumHuR6cDR.c:(.text+0xa): undefined reference to `foo'
ERROR!
collect2: error: ld returned 1 exit status
```

REF: https://gcc.gnu.org/onlinedocs/gcc/Inline.html

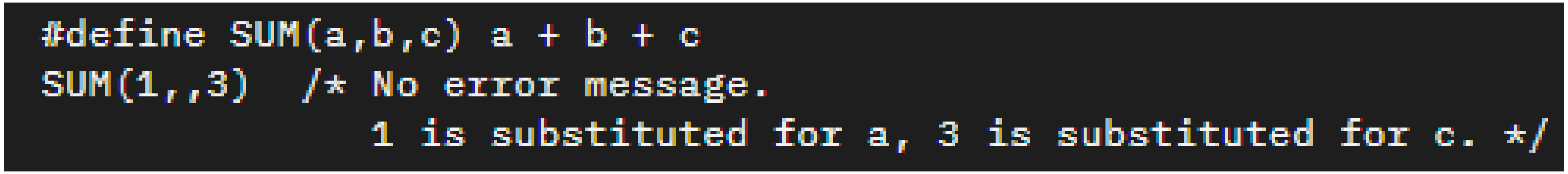
FUNCTION: NOINLINE FUNCTION ATTRIBUTE

Function attribute **noinline** prevents a function from being considered for inlining. It also disables some other interprocedural optimizations; it’s preferable to use the more comprehensive noipa attribute instead if that is your goal.

FUNCTION: FUNCTION-LIKE MACROS

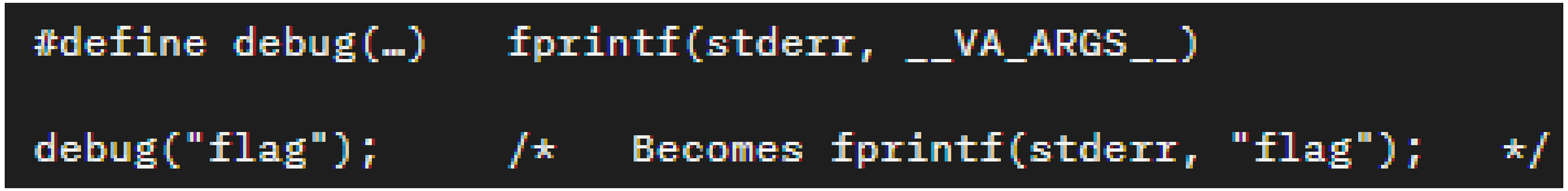
These macros are the same as a function call. It replaces the entire code instead of a function name. Pair of parentheses immediately after the macro name is necessary.

If we put a space between the macro name and the parentheses in the macro definition, then the macro will not work.



```
#define SUM(a,b,c) a + b + c
SUM(1,,3)  /* No error message.
              1 is substituted for a, 3 is substituted for c. */
```

With variable argument:



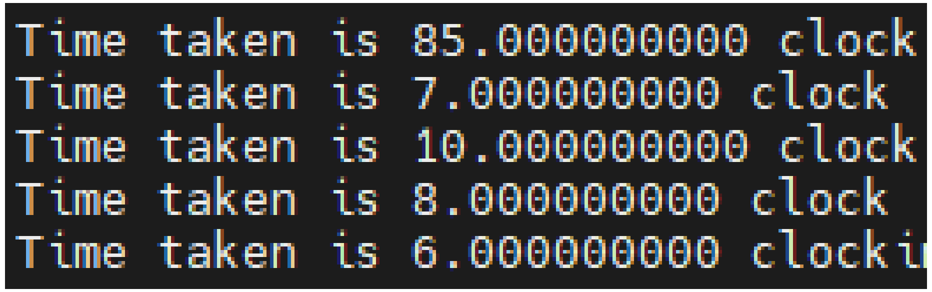
```
#define debug(…)   fprintf(stderr, __VA_ARGS__)

debug("flag");    /*   Becomes fprintf(stderr, "flag");   */
```

FUNCTION: CONST FUNCTION ATTRIBUTE

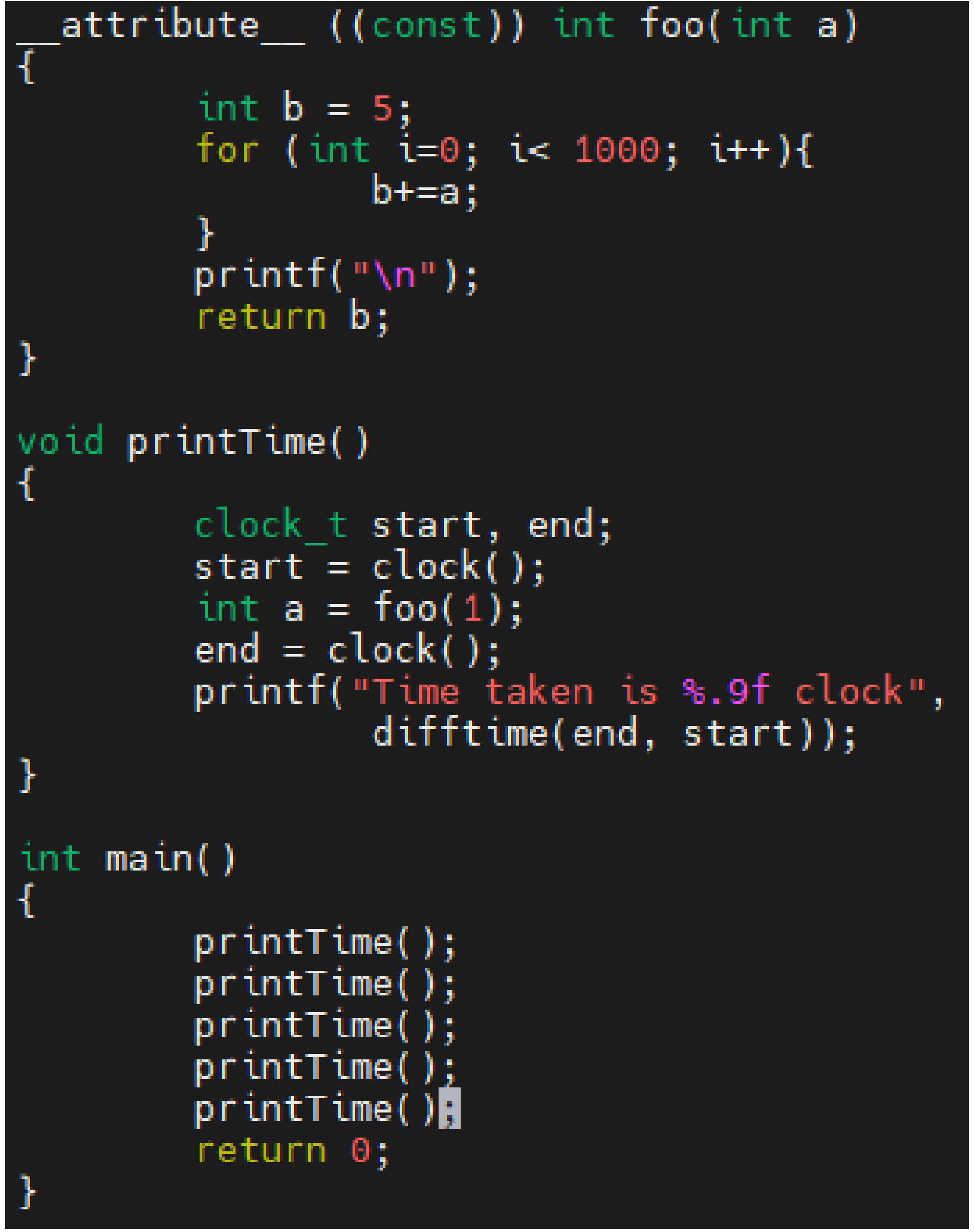
Declaring functions with the **const attribute** allows GCC to avoid emitting some calls in repeated invocations of the function with the same argument values.

For example:

Const attribute tells GCC that subsequent calls to function **foo** with the same argument value **can be replaced by the result of the first call** regardless of the statements in between.



```
Time taken is 85.000000000 clock
Time taken is 7.000000000 clock
Time taken is 10.000000000 clock
Time taken is 8.000000000 clock
Time taken is 6.000000000 clocki
```



```
__attribute__ ((const)) int foo(int a)
{
        int b = 5;
        for (int i=0; i< 1000; i++){
                b+=a;
        }
        printf("\n");
        return b;
}

void printTime()
{
        clock_t start, end;
        start = clock();
        int a = foo(1);
        end = clock();
        printf("Time taken is %.9f clock",
                difftime(end, start));
}

int main()
{
        printTime();
        printTime();
        printTime();
        printTime();
        printTime();
        return 0;
}
```

FUNCTION: ALIGNED FUNCTION ATTRIBUTE

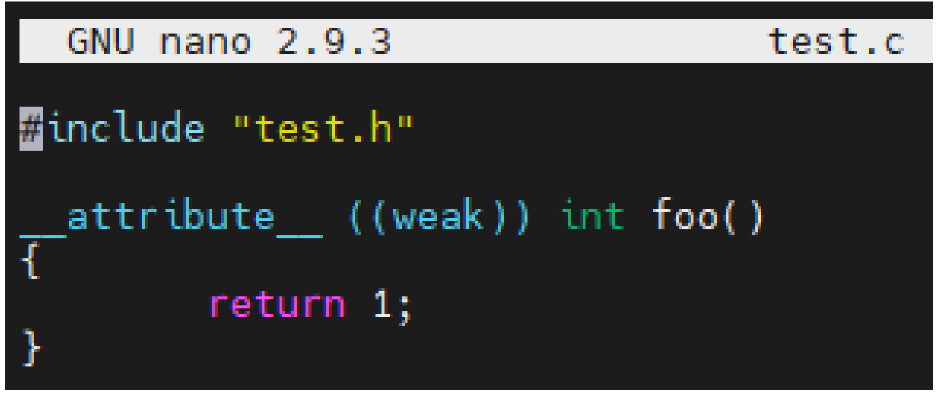
Aligned attribute specifies a **minimum alignment (in bytes)** for variables of the specified type.

Alignment is crucial for efficient memory access, especially when dealing with SIMD (Single Instruction, Multiple Data) instructions or hardware that imposes penalties for unaligned access.

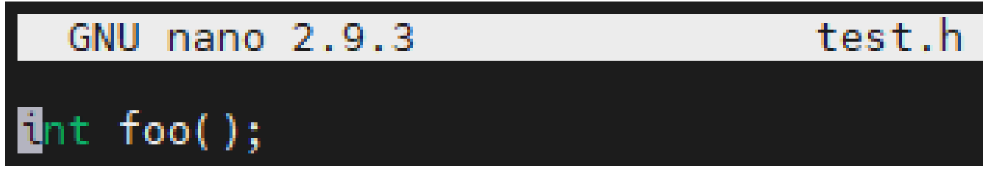
FUNCTION: WEAK FUNCTION ATTRIBUTE

The **weak function attribute** causes the symbol resulting from the function declaration to appear in the object file as a weak symbol, rather than a global one. The language feature provides the programmer writing library functions with a way to **allow function definitions in user code to override the library function declaration without causing duplicate name errors.**

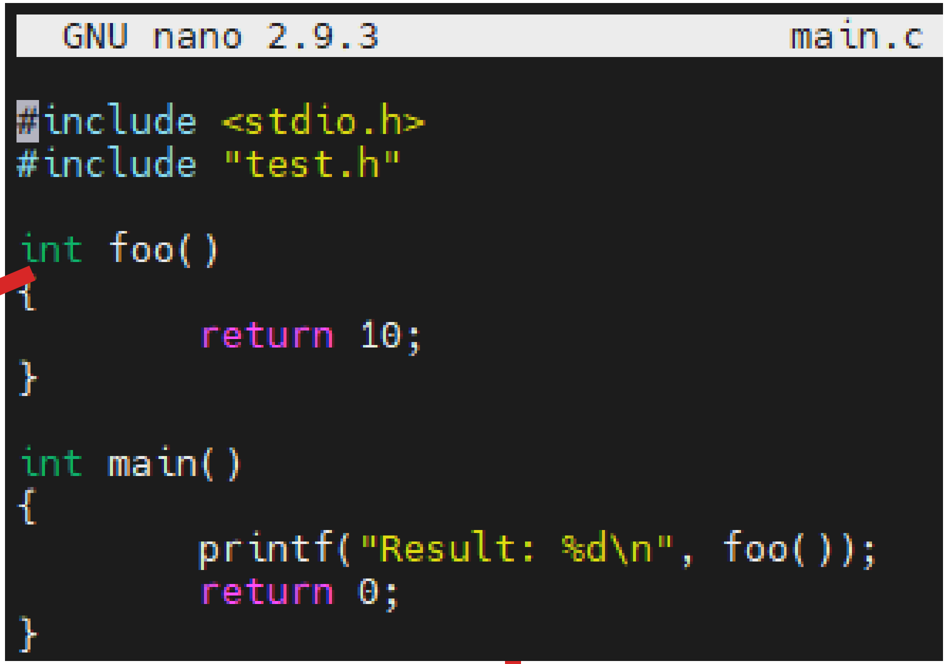
FUNCTION: WEAK FUNCTION ATTRIBUTE



```
  GNU nano 2.9.3                    test.c

#include "test.h"

__attribute__ ((weak)) int foo()
{
        return 1;
}
```

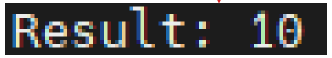


```
  GNU nano 2.9.3                    test.h

int foo();
```

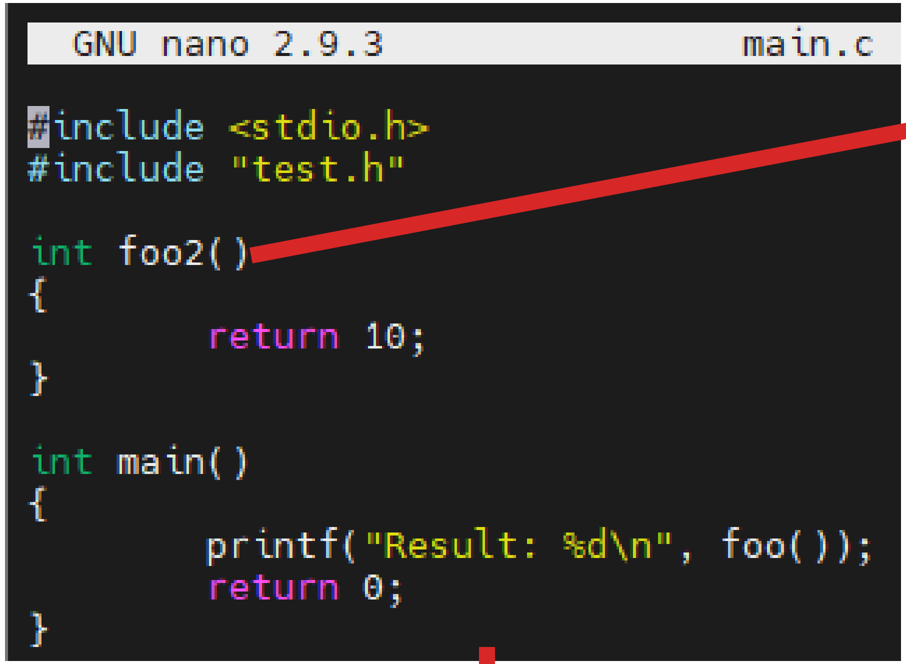


```
  GNU nano 2.9.3                    main.c

#include <stdio.h>
#include "test.h"

int foo()
{
        return 10;
}

int main()
{
        printf("Result: %d\n", foo());
        return 0;
}
```

Duplicate declare function

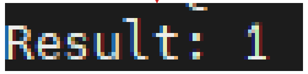


```
Result: 10
```



```
  GNU nano 2.9.3                    main.c

#include <stdio.h>
#include "test.h"

int foo2()
{
        return 10;
}

int main()
{
        printf("Result: %d\n", foo());
        return 0;
}
```

Not duplicate declare function

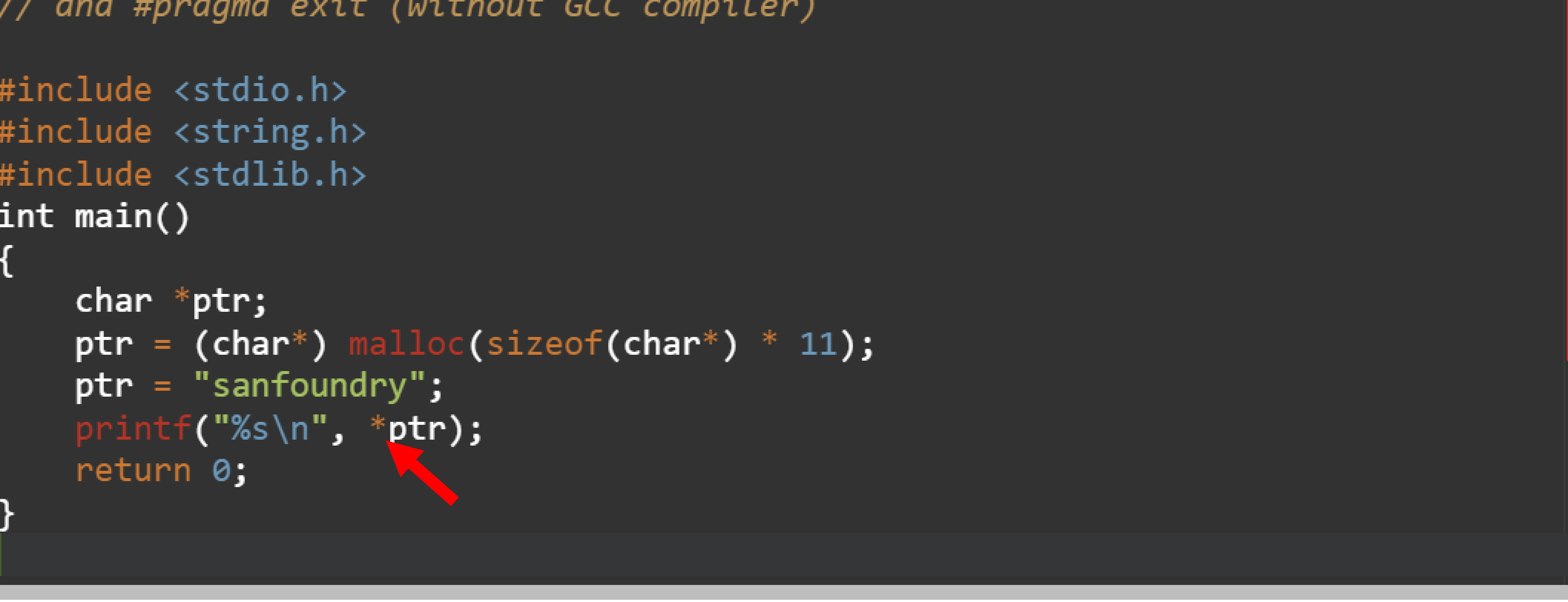


```
Result: 1
```

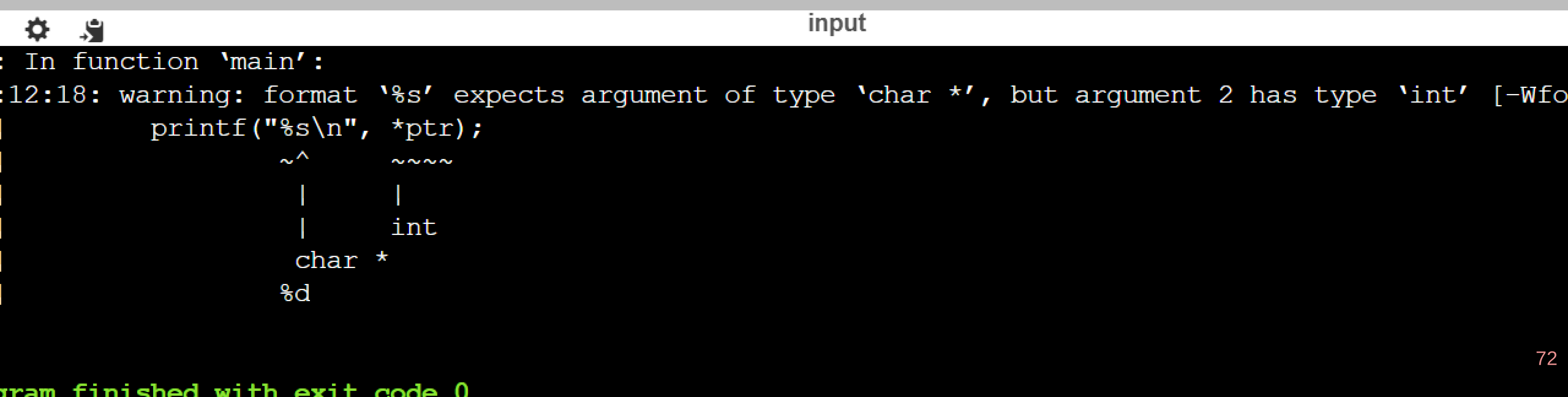
5. STRUCT, ENUM, UNION

6. SUMMARY

Theo cách của bạn



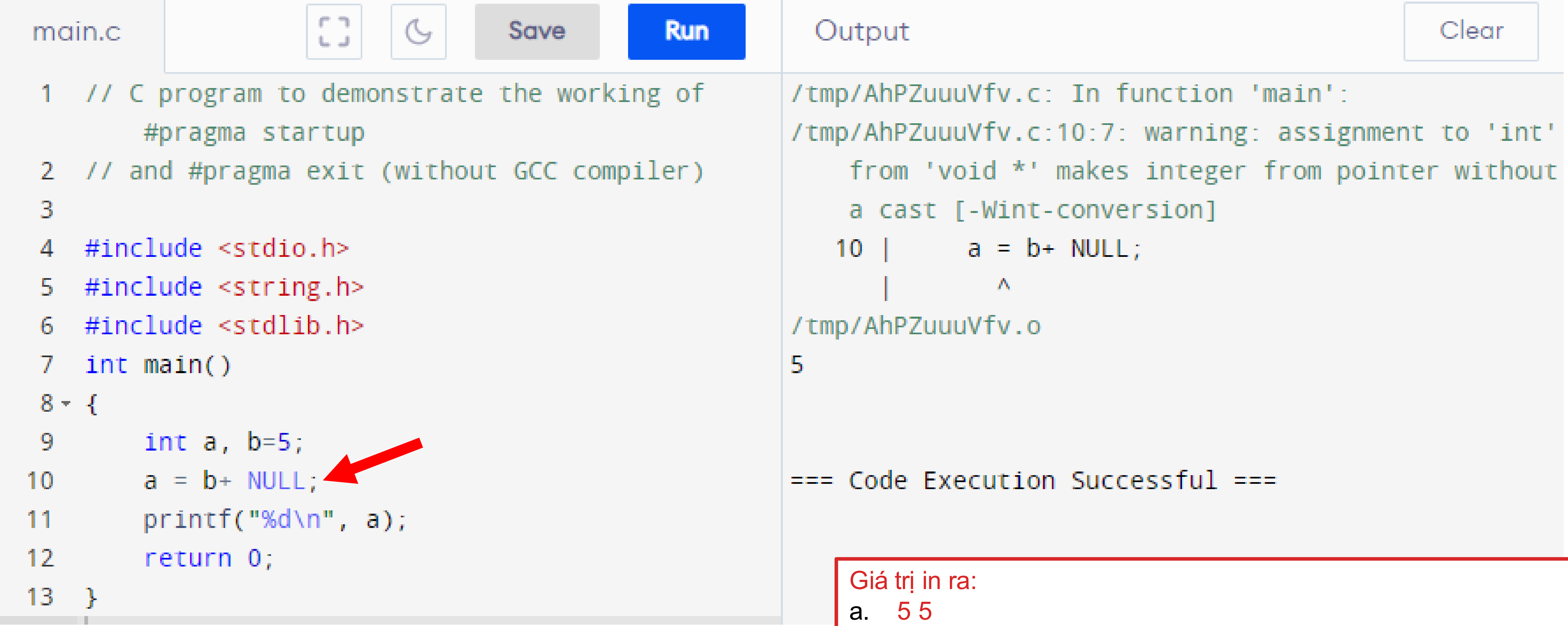
```
// and #pragma exit (without GCC compiler)

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
int main()
{
    char *ptr;
    ptr = (char*) malloc(sizeof(char*) * 11);
    ptr = "sanfoundry";
    printf("%s\n", *ptr);
    return 0;
}
```



```
: In function 'main':
:12:18: warning: format '%s' expects argument of type 'char *', but argument 2 has type 'int' [-Wfo
|          printf("%s\n", *ptr);
|                  ~^     ~~~~
|                   |     |
|                   |     int
|                   char *
|                  %d


gram finished with exit code 0
```

a. Sanfoundry
b. Segment fault
c. Khong co cau trl dung
d. Syntax error

Giải thích: in ra *ptr là việ
cập vào vùng nhớ có địa c
> segment fault



```c
// C program to demonstrate the working of #pragma startup
// and #pragma exit (without GCC compiler)

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
int main()
{
    int a, b=5;
    a = b+ NULL;
    printf("%d\n", a);
    return 0;
}
```

```
/tmp/AhPZuuuVfv.c: In function 'main':
/tmp/AhPZuuuVfv.c:10:7: warning: assignment to 'int'
    from 'void *' makes integer from pointer without
    a cast [-Wint-conversion]
   10 |     a = b+ NULL;
      |       ^
/tmp/AhPZuuuVfv.o
5


=== Code Execution Successful ===
```

Giá trị in ra:

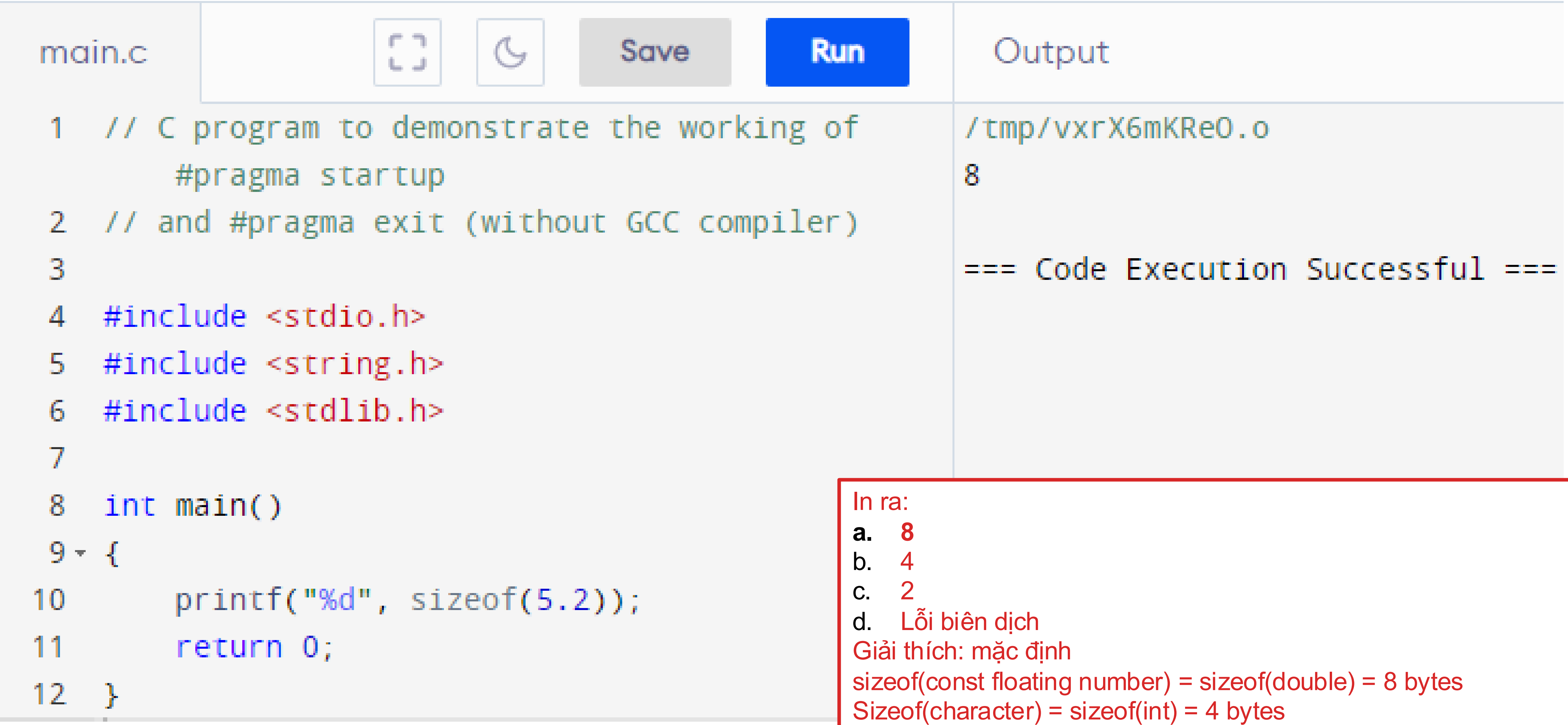
a. 5 5

b. 5

c. Không in ra

d. 6

Giải thích: NULL là con trỏ void, có giá trị bằng 0 khi ép kiểu sang int



```c
// C program to demonstrate the working of #pragma startup
// and #pragma exit (without GCC compiler)

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int main()
{
    printf("%d", sizeof(5.2));
    return 0;
}
```

```
/tmp/vxrX6mKReO.o
8


=== Code Execution Successful ===
```

In ra:

a. **8**

b. 4

c. 2

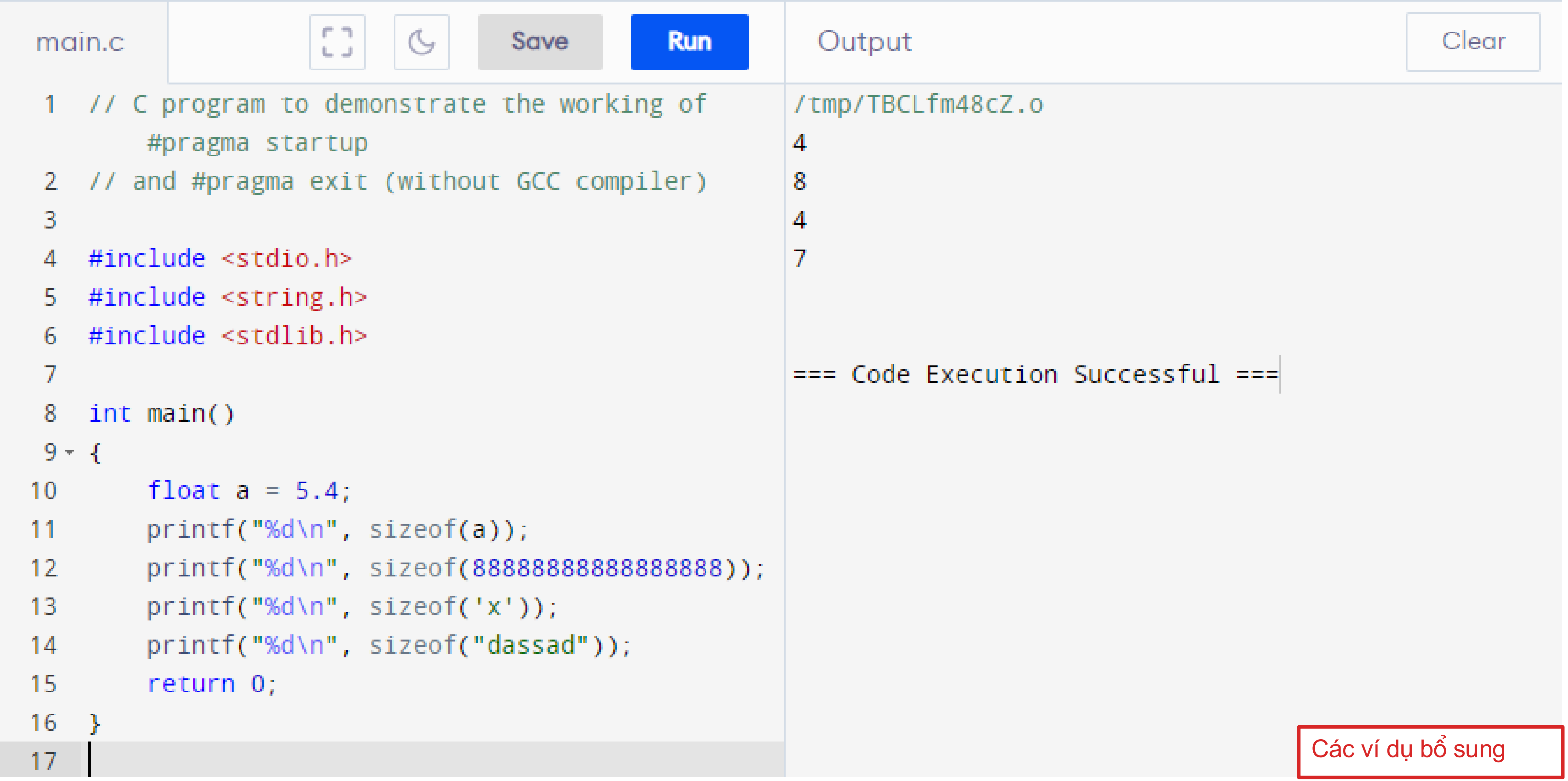
d. Lỗi biên dịch

Giải thích: mặc định

sizeof(const floating number) = sizeof(double) = 8 bytes

Sizeof(character) = sizeof(int) = 4 bytes

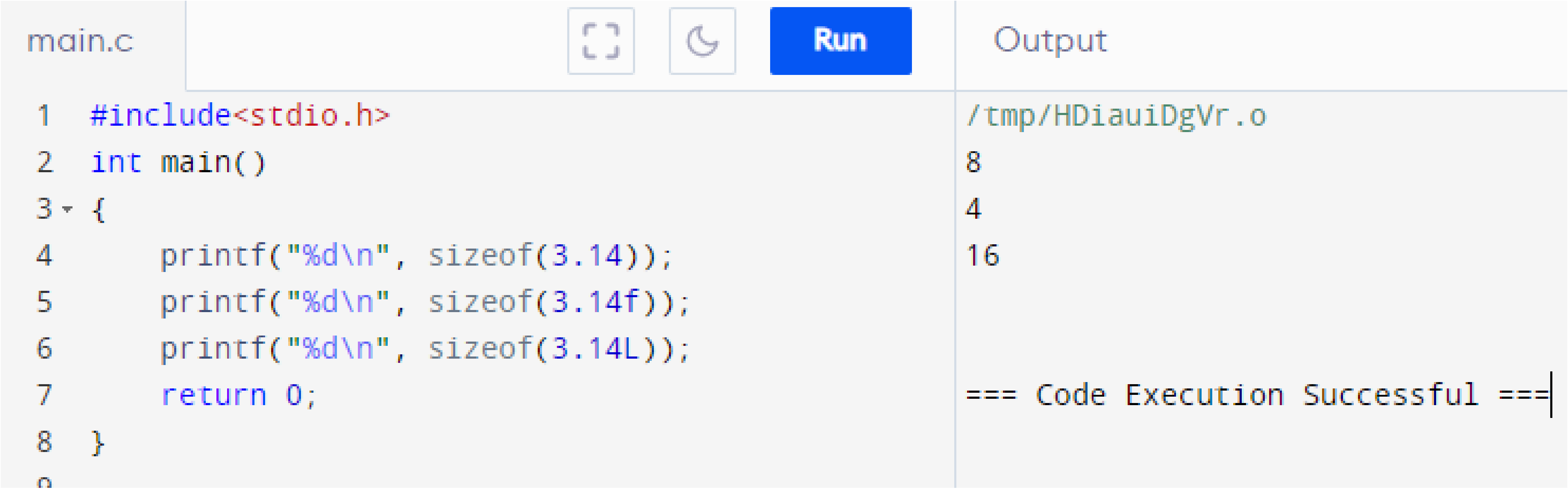
Sizeof(int too long) = sizeof(long) or sizeof(long long) = 8 or 16



```c
// C program to demonstrate the working of
    #pragma startup
// and #pragma exit (without GCC compiler)

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int main()
{
    float a = 5.4;
    printf("%d\n", sizeof(a));
    printf("%d\n", sizeof(88888888888888888));
    printf("%d\n", sizeof('x'));
    printf("%d\n", sizeof("dassad"));
    return 0;
}
```

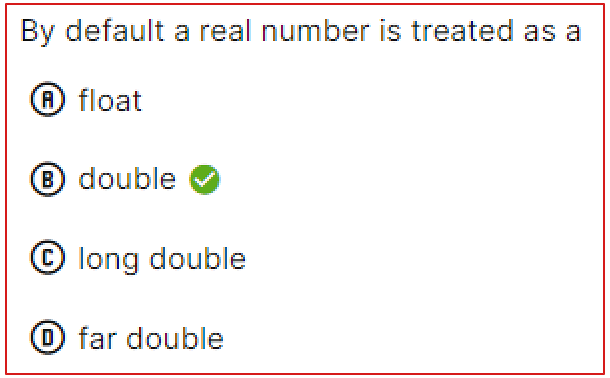
```
/tmp/TBCLfm48cZ.o
4
8
4
7


=== Code Execution Successful ===
```

Các ví dụ bổ sung



```c
#include<stdio.h>
int main()
{
    printf("%d\n", sizeof(3.14));
    printf("%d\n", sizeof(3.14f));
    printf("%d\n", sizeof(3.14L));
    return 0;
}
```

```
/tmp/HDiauiDgVr.o
8
4
16


=== Code Execution Successful ===
```



By default a real number is treated as a

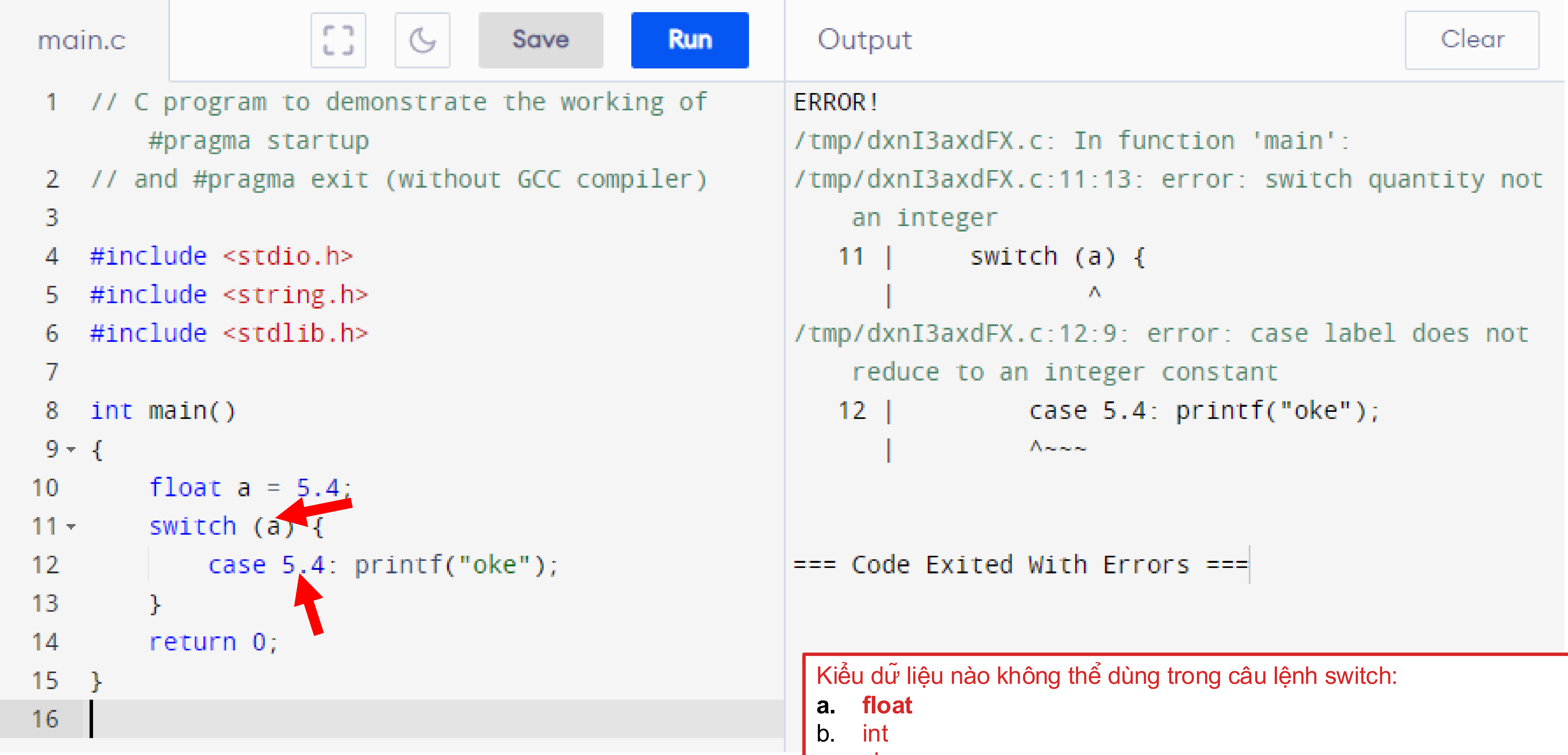
Ⓐ float

Ⓑ double ✓

Ⓒ long double

Ⓓ far double

Các ví dụ bổ sung



```c
// C program to demonstrate the working of #pragma startup
// and #pragma exit (without GCC compiler)

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int main()
{
    float a = 5.4;
    switch (a) {
        case 5.4: printf("oke");
    }
    return 0;
}
```

```
ERROR!
/tmp/dxnI3axdFX.c: In function 'main':
/tmp/dxnI3axdFX.c:11:13: error: switch quantity not
    an integer
   11 |     switch (a) {
      |             ^
/tmp/dxnI3axdFX.c:12:9: error: case label does not
    reduce to an integer constant
   12 |         case 5.4: printf("oke");
      |         ^~~~


=== Code Exited With Errors ===
```

Kiểu dữ liệu nào không thể dùng trong câu lệnh switch:

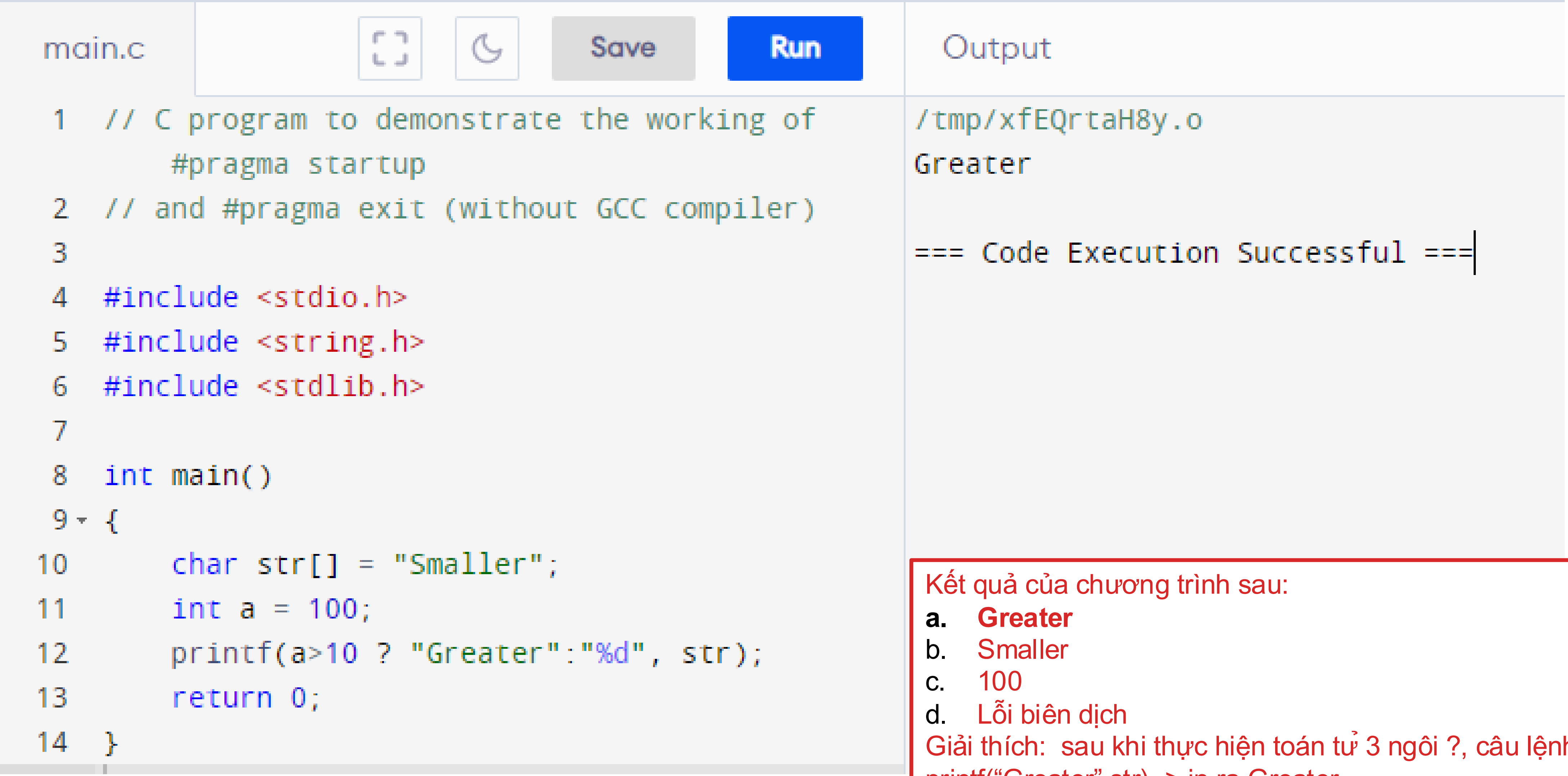
a. float

b. int

c. char

d. enum

Giải thích: ví dụ trên



```c
// C program to demonstrate the working of
    #pragma startup
// and #pragma exit (without GCC compiler)

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int main()
{
    char str[] = "Smaller";
    int a = 100;
    printf(a>10 ? "Greater":"%d", str);
    return 0;
}
```

```
/tmp/xfEQrtaH8y.o
Greater


=== Code Execution Successful ===
```

Kết quả của chương trình sau:

a. **Greater**

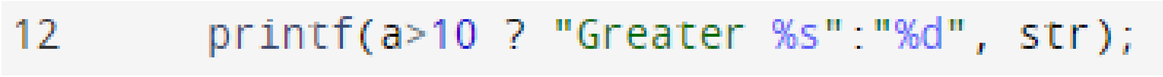
b. Smaller

c. 100

d. Lỗi biên dịch

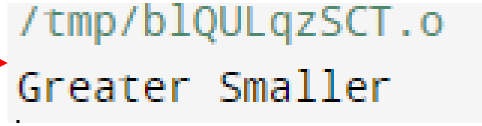
Giải thích: sau khi thực hiện toán tử 3 ngôi ?, câu lệnh tương đương với printf(“Greater”,str) -> in ra Greater

Chứng minh:

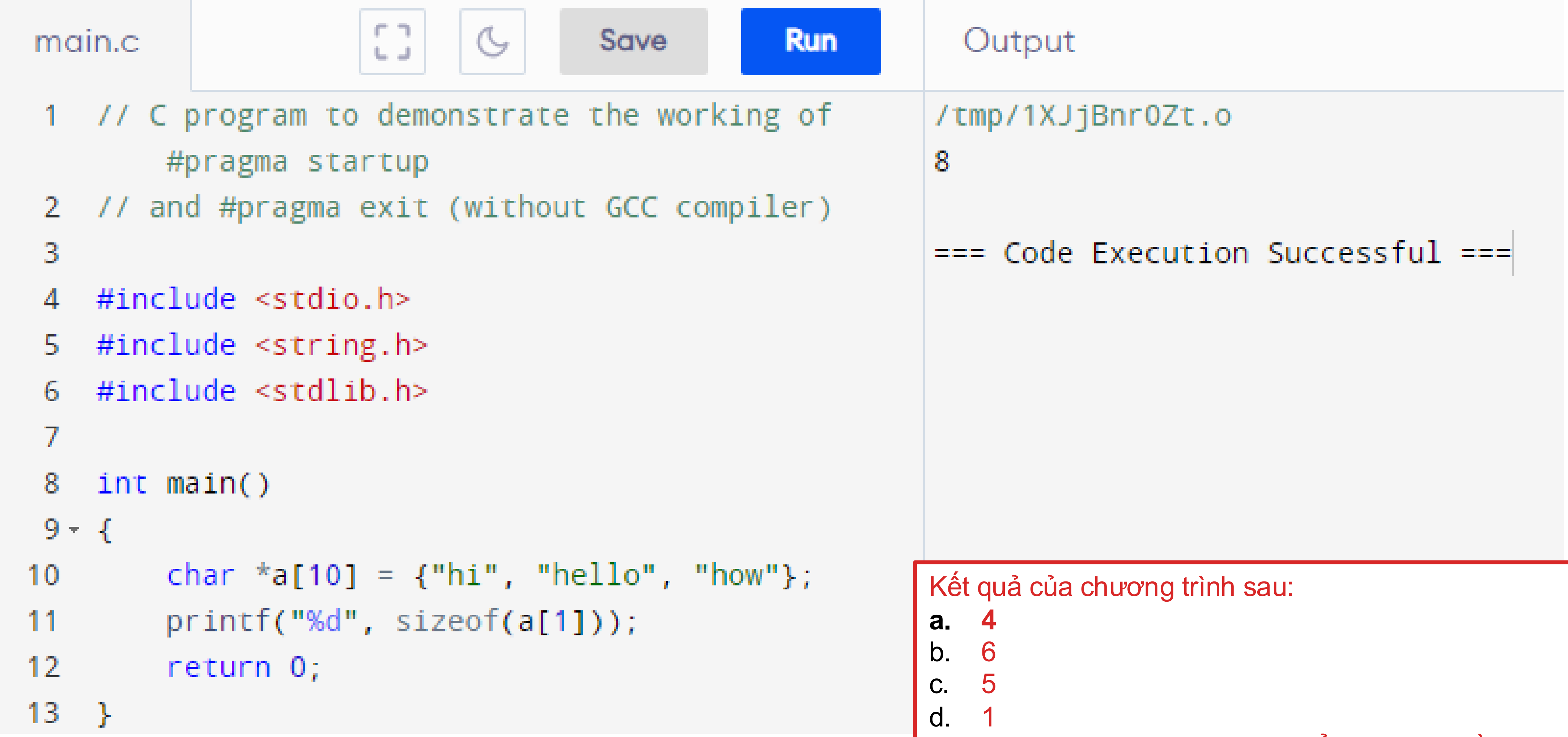


```c
    printf(a>10 ? "Greater %s":"%d", str);
```

→



```
/tmp/blQULqzSCT.o
Greater Smaller
```



```c
// C program to demonstrate the working of #pragma startup
// and #pragma exit (without GCC compiler)

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int main()
{
    char *a[10] = {"hi", "hello", "how"};
    printf("%d", sizeof(a[1]));
    return 0;
}
```

```
/tmp/1XJjBnr0Zt.o
8


=== Code Execution Successful ===
```

Kết quả của chương trình sau:

a. **4**

b. 6

c. 5

d. 1

Giải thích: a[1] là 1 địa chỉ, có thể in ra hello từ a[1]. Địa chỉ có thể bằng 4 bytes với 32 bit và 8 bytes với 64 bit.

Trong điều kiện bài toán này, chỉ có thể chọn 4

Lựa chọn nào cho phép liên kết file thư viện trong gcc:

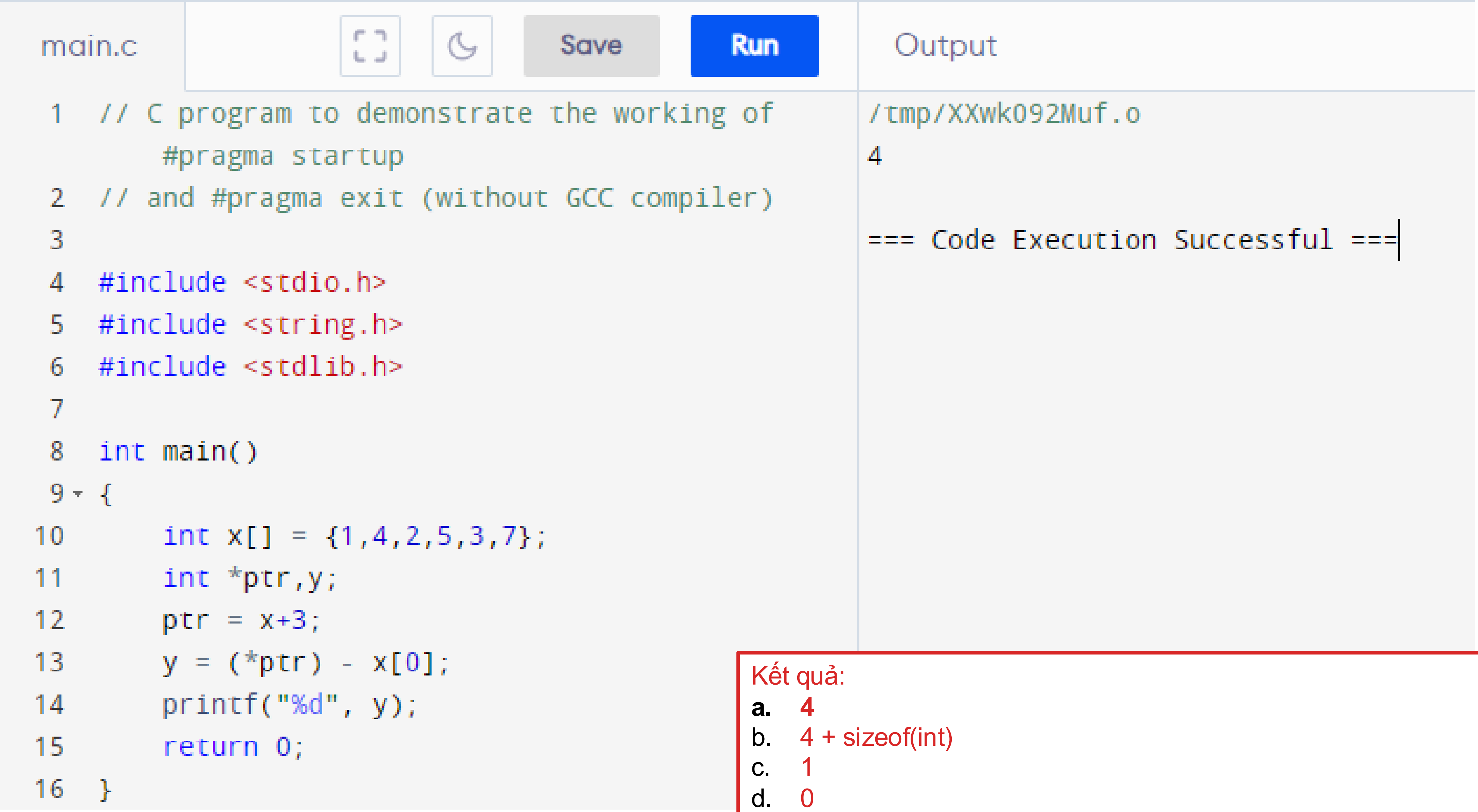
a. -L
b. -l
c. -link
d. Không có

Giải thích:

-L [addr] là liên kết các file thư viện nằm tại [addr]

-l chỉ đơn giản là cách viết tương đương: -lfun ~ libfun.so ~ libfun.dll

-link : không tồn tại



```c
// C program to demonstrate the working of
    #pragma startup
// and #pragma exit (without GCC compiler)

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int main()
{
    int x[] = {1,4,2,5,3,7};
    int *ptr,y;
    ptr = x+3;
    y = (*ptr) - x[0];
    printf("%d", y);
    return 0;
}
```

```
/tmp/XXwk092Muf.o
4

=== Code Execution Successful ===
```

Kết quả:

a. **4**

b. 4 + sizeof(int)

c. 1

d. 0

Giải thích:

Trong 1 chương trình C++ phải có:

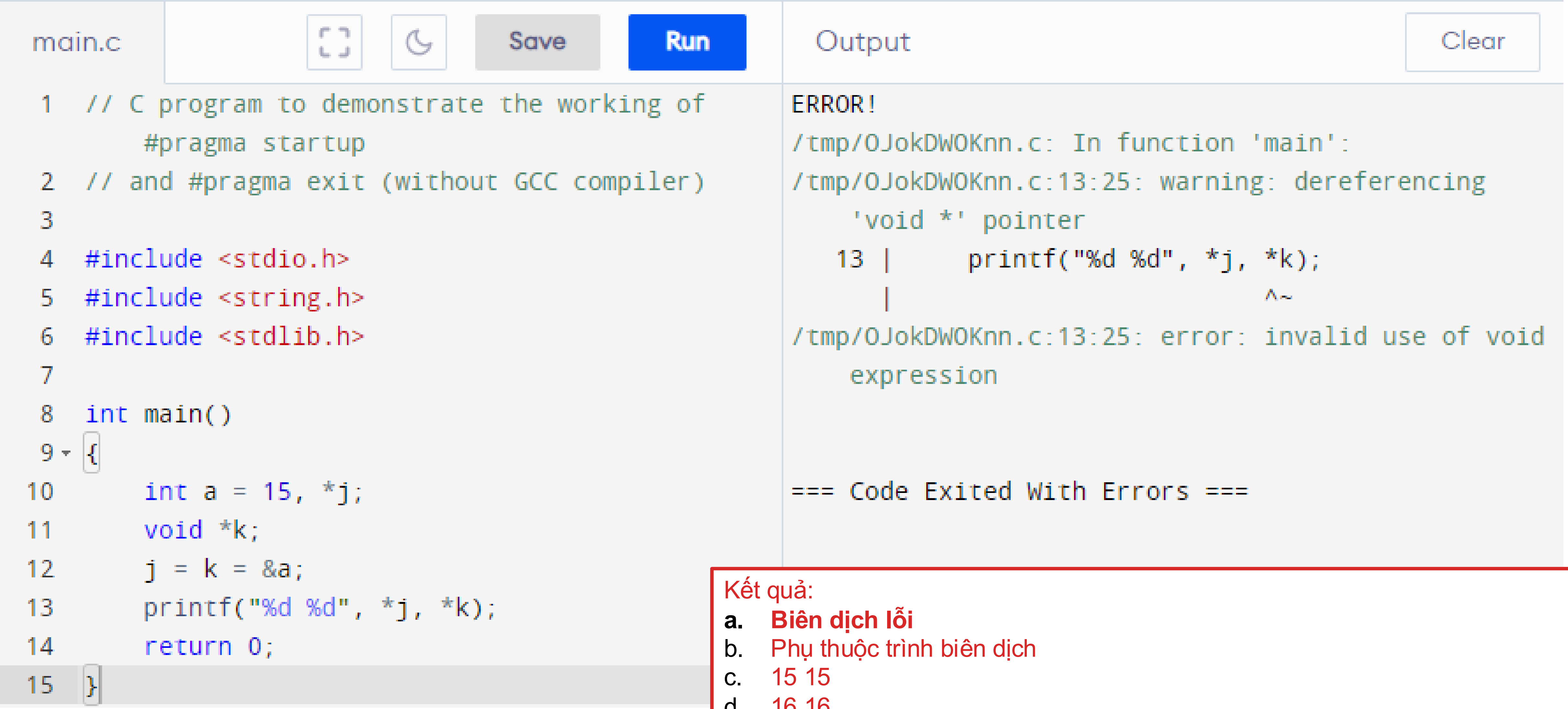
a. Hàm main

b. Kiểu dữ liệu trả về của hàm main

c. Marco #include

d. Biến toàn cục

Giải thích:



```c
// C program to demonstrate the working of #pragma startup
// and #pragma exit (without GCC compiler)

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int main()
{
    int a = 15, *j;
    void *k;
    j = k = &a;
    printf("%d %d", *j, *k);
    return 0;
}
```

```
ERROR!
/tmp/OJokDWOKnn.c: In function 'main':
/tmp/OJokDWOKnn.c:13:25: warning: dereferencing
    'void *' pointer
   13 |     printf("%d %d", *j, *k);
      |                         ^~
/tmp/OJokDWOKnn.c:13:25: error: invalid use of void
    expression


=== Code Exited With Errors ===
```

Kết quả:

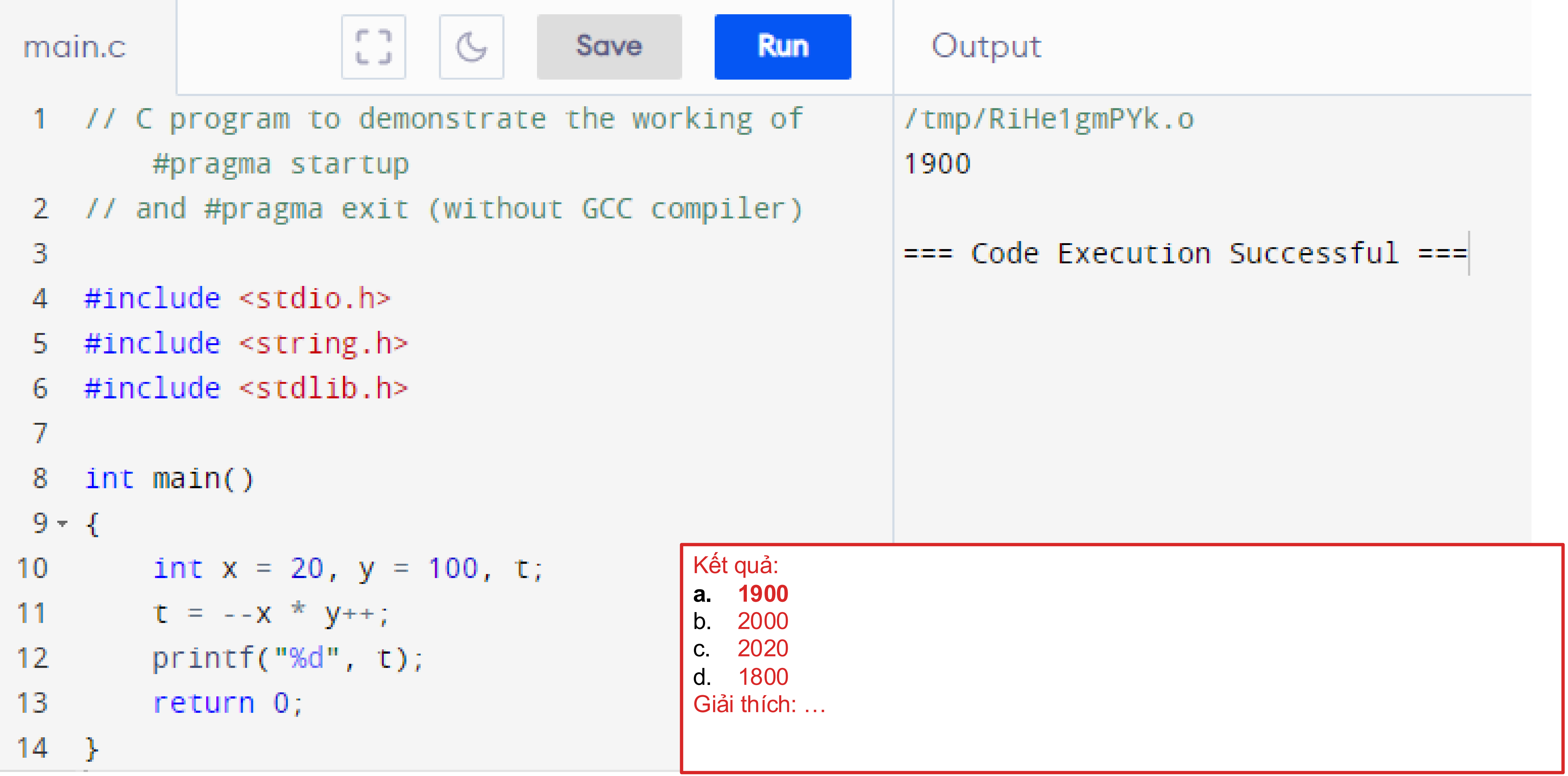
a. **Biên dịch lỗi**

b. Phụ thuộc trình biên dịch

c. 15 15

d. 16 16

Giải thích: lấy giá trị từ con trỏ void mà không ép kiểu



```c
// C program to demonstrate the working of #pragma startup
// and #pragma exit (without GCC compiler)

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int main()
{
    int x = 20, y = 100, t;
    t = --x * y++;
    printf("%d", t);
    return 0;
}
```

```
/tmp/RiHe1gmPYk.o
1900


=== Code Execution Successful ===
```

Kết quả:

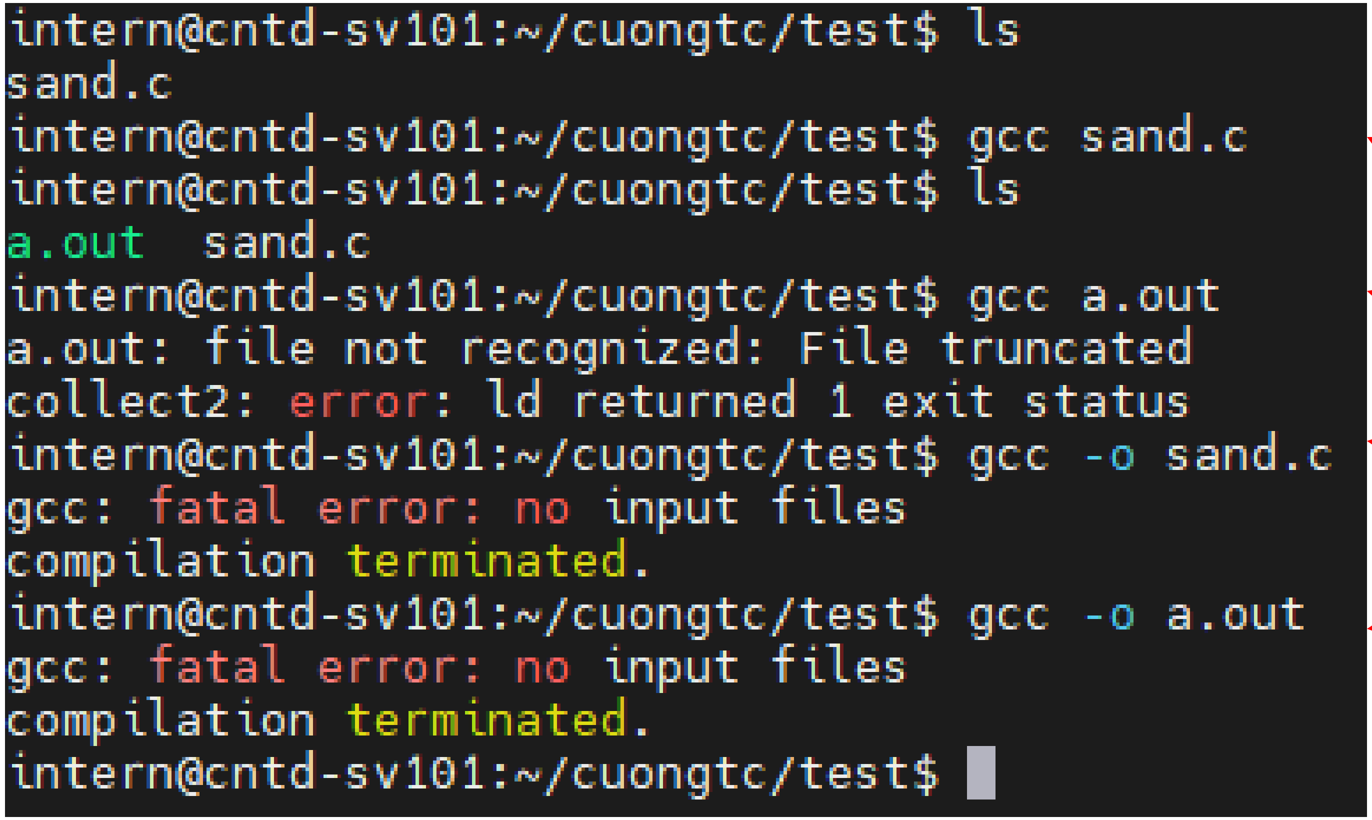
a. **1900**

b. 2000

c. 2020

d. 1800

Giải thích: …



```
intern@cntd-sv101:~/cuongtc/test$ ls
sand.c
intern@cntd-sv101:~/cuongtc/test$ gcc sand.c
intern@cntd-sv101:~/cuongtc/test$ ls
a.out  sand.c
intern@cntd-sv101:~/cuongtc/test$ gcc a.out
a.out: file not recognized: File truncated
collect2: error: ld returned 1 exit status
intern@cntd-sv101:~/cuongtc/test$ gcc -o sand.c
gcc: fatal error: no input files
compilation terminated.
intern@cntd-sv101:~/cuongtc/test$ gcc -o a.out
gcc: fatal error: no input files
compilation terminated.
intern@cntd-sv101:~/cuongtc/test$ 
```

Câu lệnh nào thực thi file a.out:

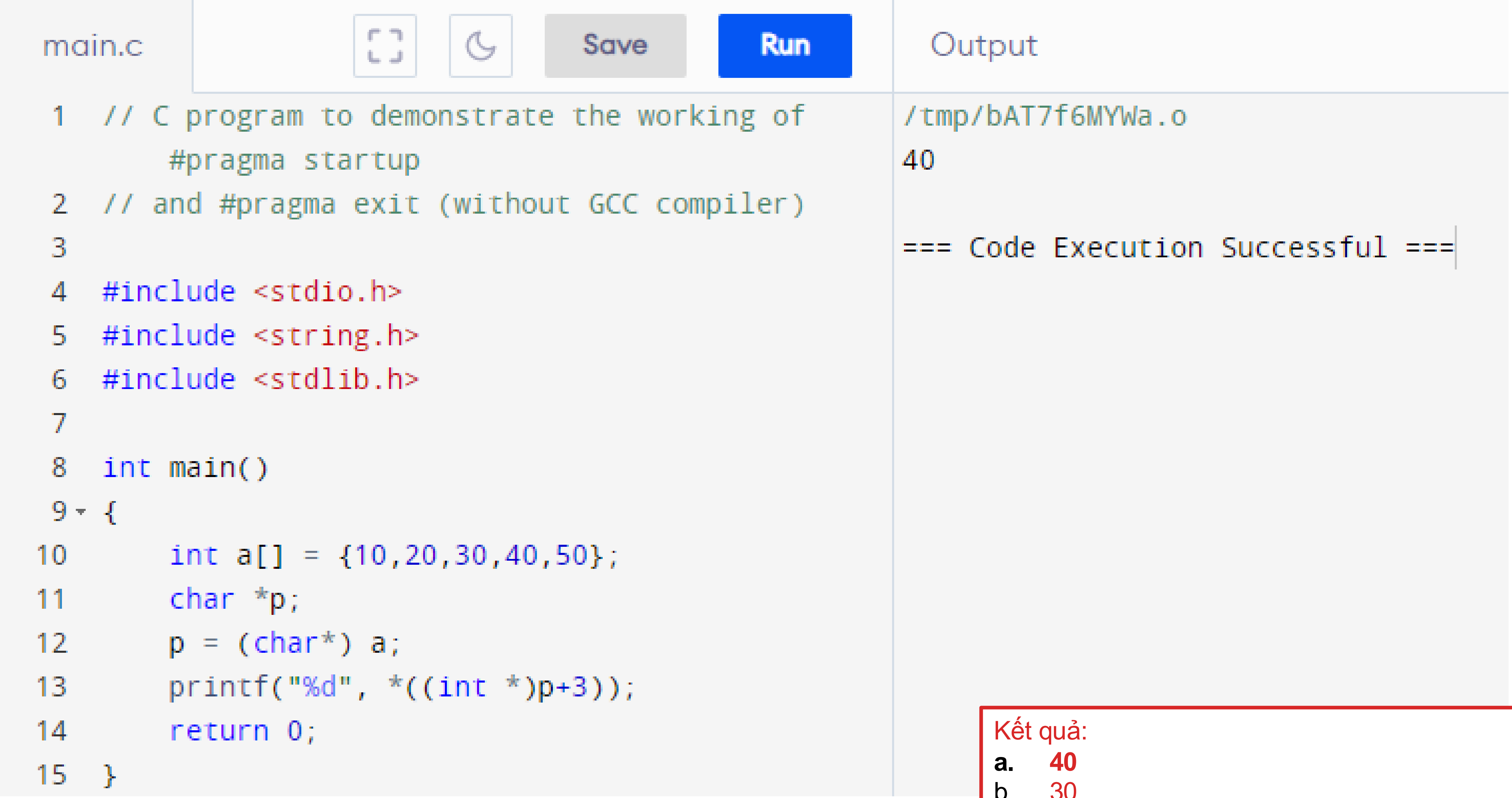
a. **gcc sand.c**

b. gcc –o sand.c

c. gcc –o a.out

d. gcc a.out

Giải thích: …



```c
// C program to demonstrate the working of #pragma startup
// and #pragma exit (without GCC compiler)

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int main()
{
    int a[] = {10,20,30,40,50};
    char *p;
    p = (char*) a;
    printf("%d", *((int *)p+3));
    return 0;
}
```

```
/tmp/bAT7f6MYWa.o
40


=== Code Execution Successful ===
```

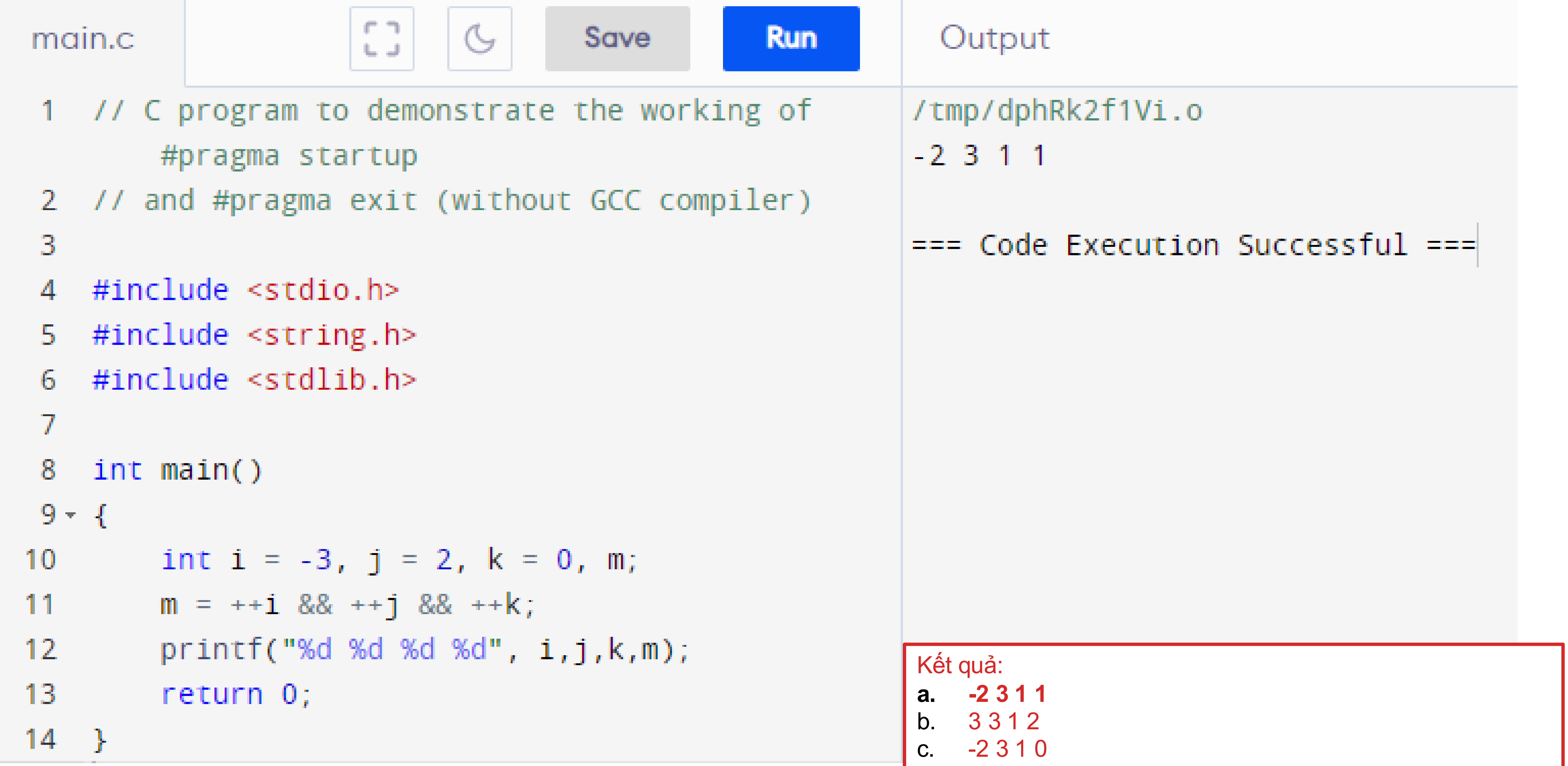
Kết quả:

a. **40**

b. 30

c. 20

d. 50



```c
// C program to demonstrate the working of
    #pragma startup
// and #pragma exit (without GCC compiler)

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int main()
{
    int i = -3, j = 2, k = 0, m;
    m = ++i && ++j && ++k;
    printf("%d %d %d %d", i,j,k,m);
    return 0;
}
```

```
/tmp/dphRk2f1Vi.o
-2 3 1 1


=== Code Execution Successful ===
```

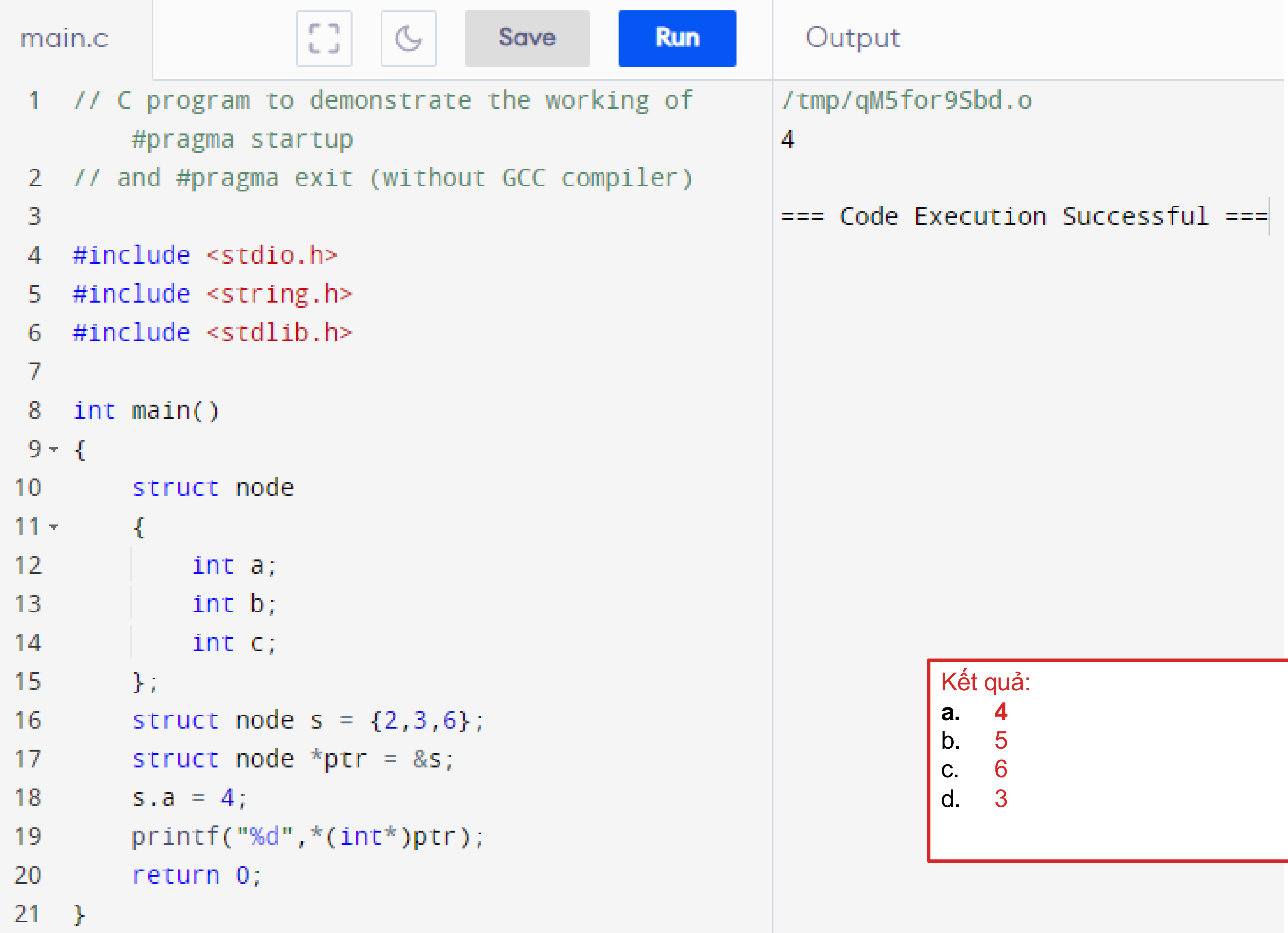
Kết quả:

a. **-2 3 1 1**

b. 3 3 1 2

c. -2 3 1 0

d. 1 2 3 1



```c
// C program to demonstrate the working of #pragma startup
// and #pragma exit (without GCC compiler)

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int main()
{
    struct node
    {
        int a;
        int b;
        int c;
    };
    struct node s = {2,3,6};
    struct node *ptr = &s;
    s.a = 4;
    printf("%d",*(int*)ptr);
    return 0;
}
```

```
/tmp/qM5for9Sbd.o
4


=== Code Execution Successful ===
```

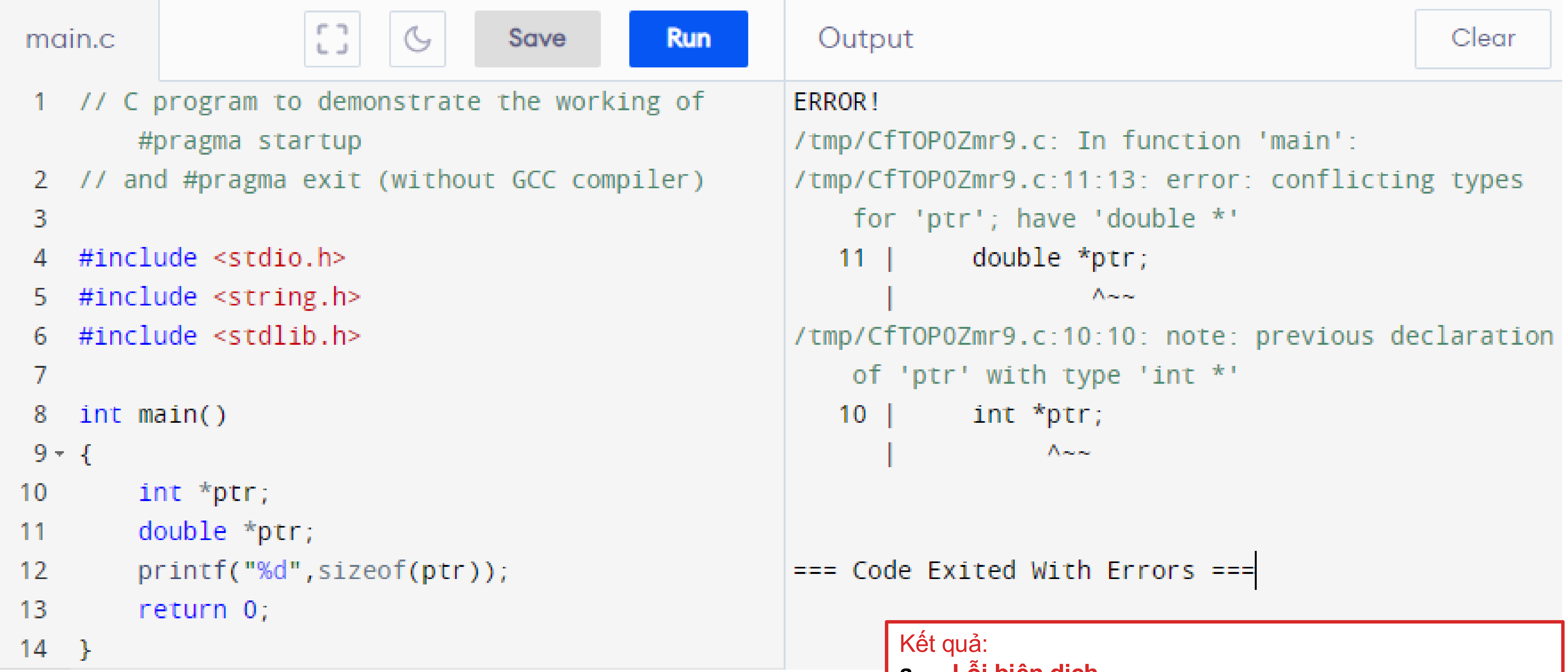
Kết quả:

a. **4**

b. 5

c. 6

d. 3



```c
// C program to demonstrate the working of #pragma startup
// and #pragma exit (without GCC compiler)

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int main()
{
    int *ptr;
    double *ptr;
    printf("%d",sizeof(ptr));
    return 0;
}
```

```
ERROR!
/tmp/CfTOP0Zmr9.c: In function 'main':
/tmp/CfTOP0Zmr9.c:11:13: error: conflicting types
    for 'ptr'; have 'double *'
   11 |     double *ptr;
      |             ^~~
/tmp/CfTOP0Zmr9.c:10:10: note: previous declaration
    of 'ptr' with type 'int *'
   10 |     int *ptr;
      |          ^~~


=== Code Exited With Errors ===
```

Kết quả:

a. **Lỗi biên dịch**

b. Segment fault

c. 4

d. 8

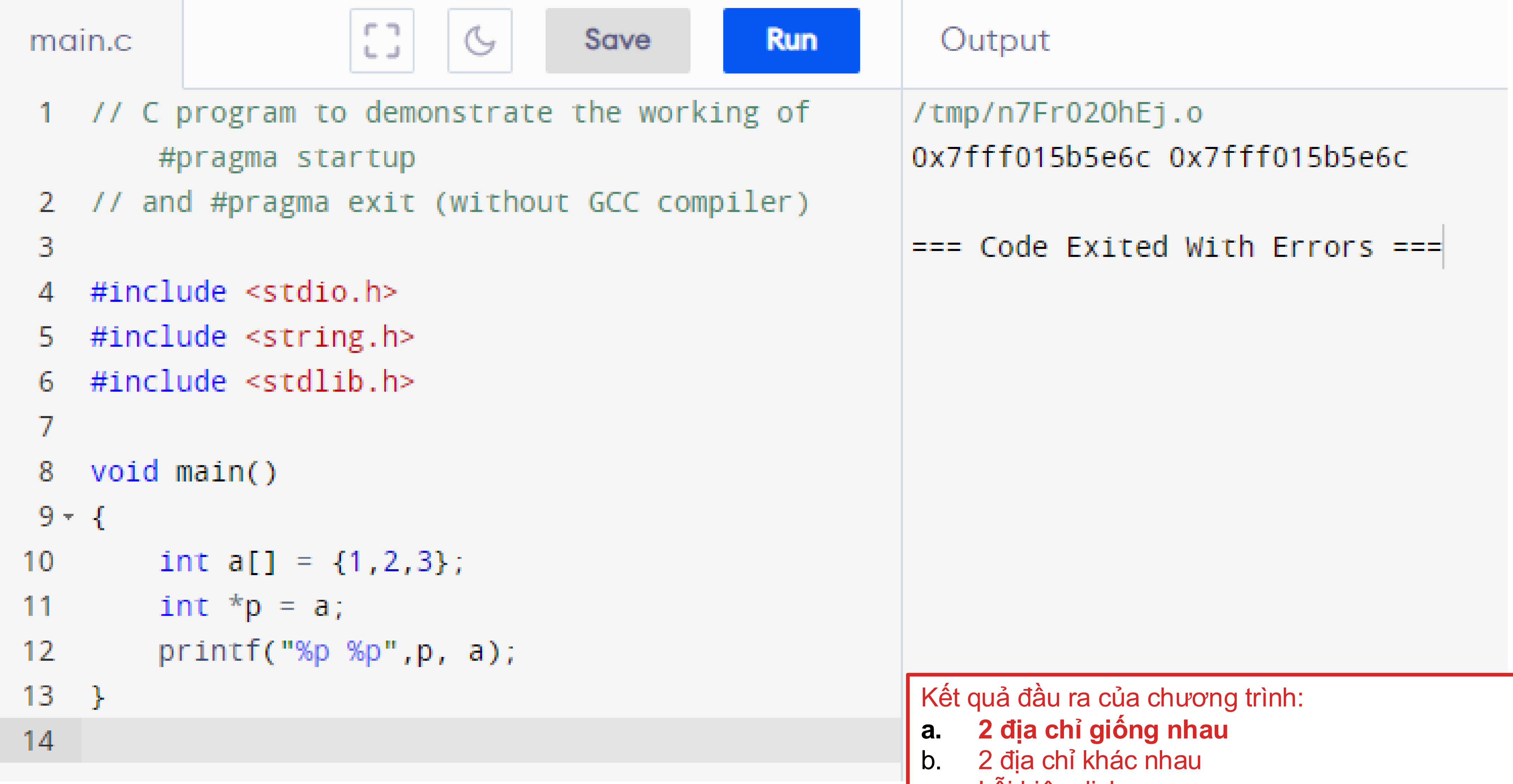
Phần khai báo biến nội bộ theo chuẩn anci C:

a. Đặt ngay sau dấu { đầu tiên của thân hàm, trước bất cứ lệnh nào khác

b. Mọi nơi

c. Ngay sau #include

d. Sau các khai báo marco



```c
// C program to demonstrate the working of #pragma startup
// and #pragma exit (without GCC compiler)

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

void main()
{
    int a[] = {1,2,3};
    int *p = a;
    printf("%p %p",p, a);
}

```

```
/tmp/n7Fr02OhEj.o
0x7fff015b5e6c 0x7fff015b5e6c

=== Code Exited With Errors ===
```

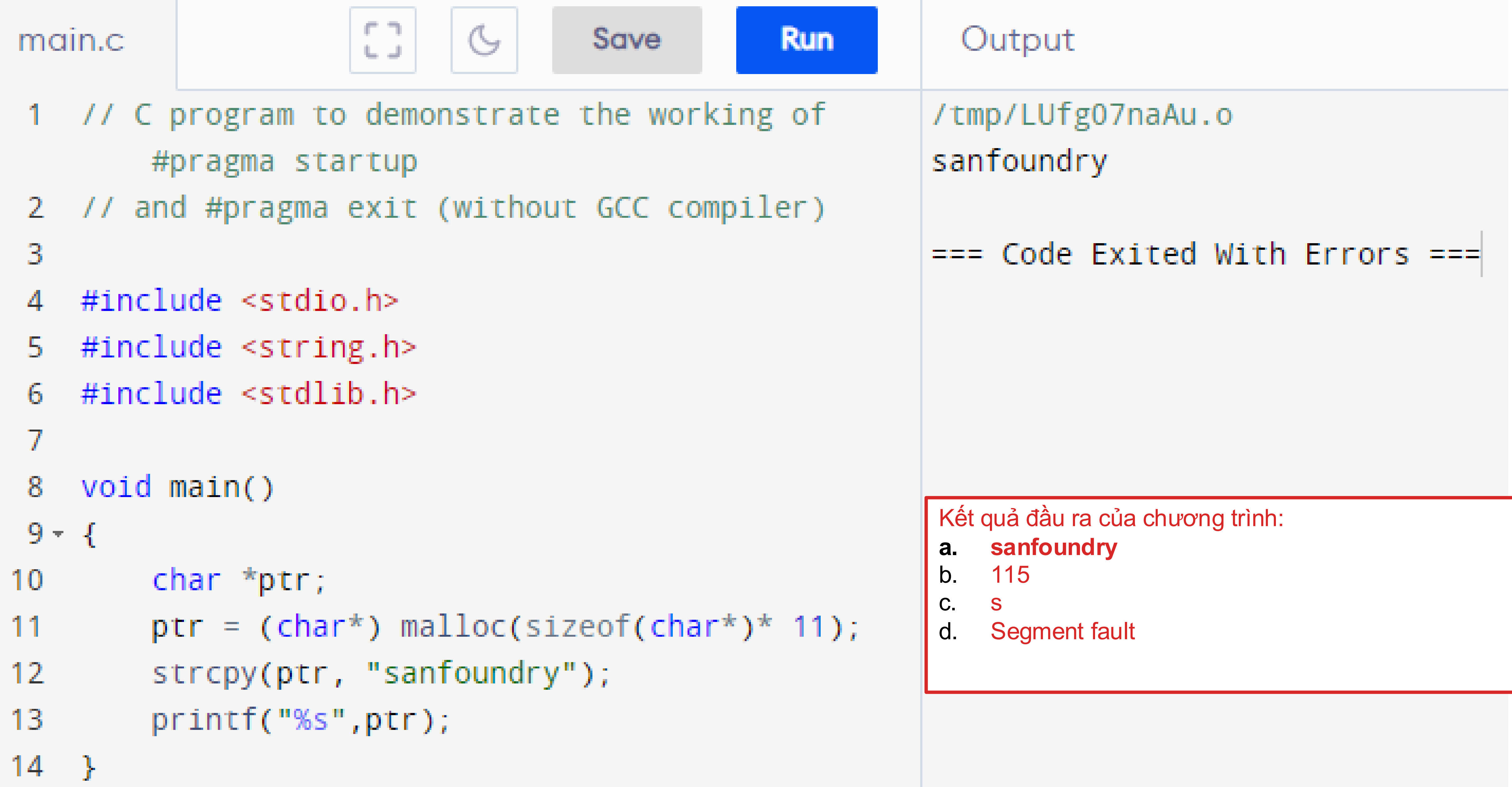
Kết quả đầu ra của chương trình:

a. **2 địa chỉ giống nhau**

b. 2 địa chỉ khác nhau

c. Lỗi biên dịch

d. Không in ra



```c
// C program to demonstrate the working of #pragma startup
// and #pragma exit (without GCC compiler)

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

void main()
{
    char *ptr;
    ptr = (char*) malloc(sizeof(char*)* 11);
    strcpy(ptr, "sanfoundry");
    printf("%s",ptr);
}
```

```
/tmp/LUfg07naAu.o
sanfoundry

=== Code Exited With Errors ===
```

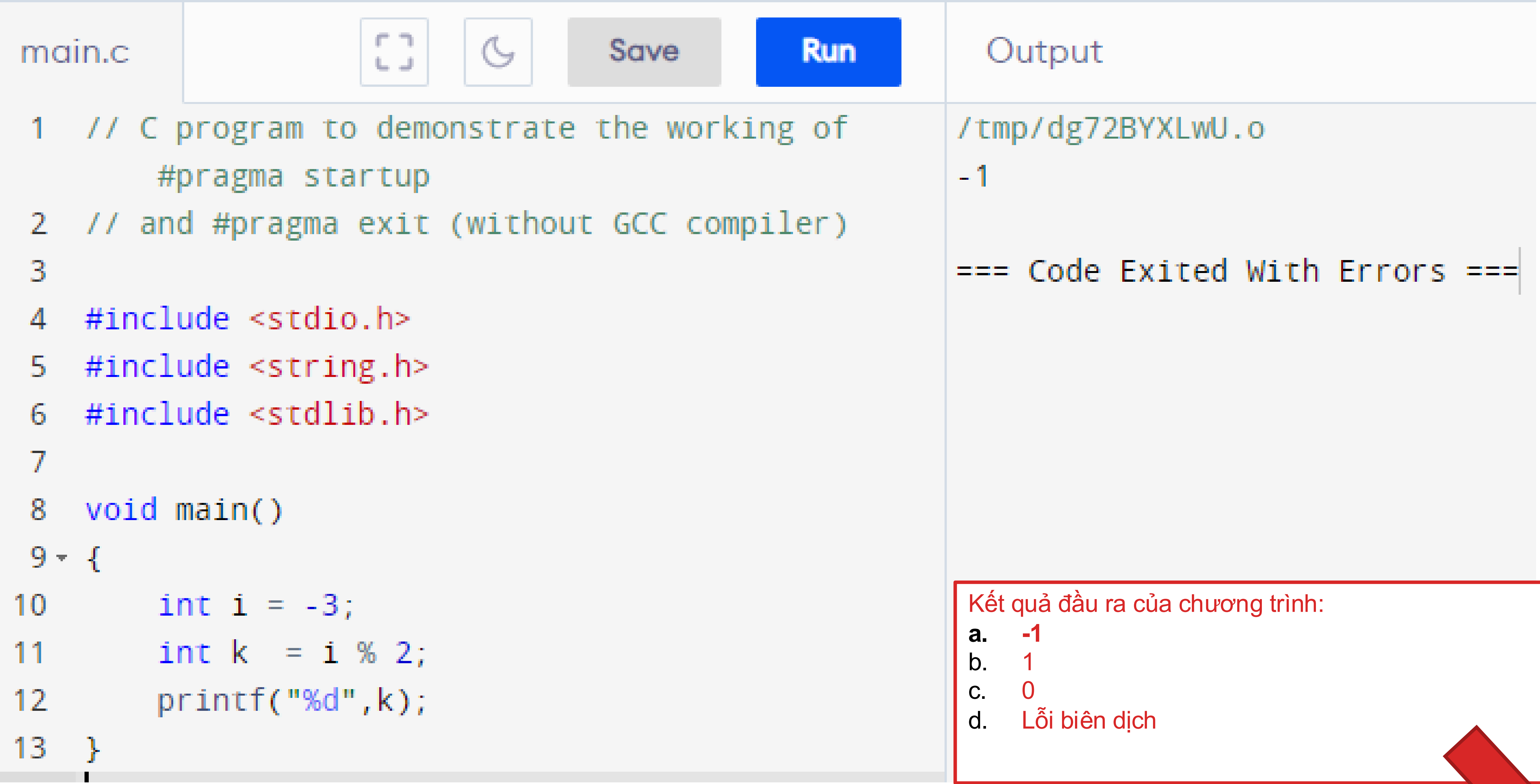
Kết quả đầu ra của chương trình:

a. sanfoundry

b. 115

c. s

d. Segment fault



```c
// C program to demonstrate the working of #pragma startup
// and #pragma exit (without GCC compiler)

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

void main()
{
    int i = -3;
    int k  = i % 2;
    printf("%d",k);
}
```

```
/tmp/dg72BYXLwU.o
-1

=== Code Exited With Errors ===
```

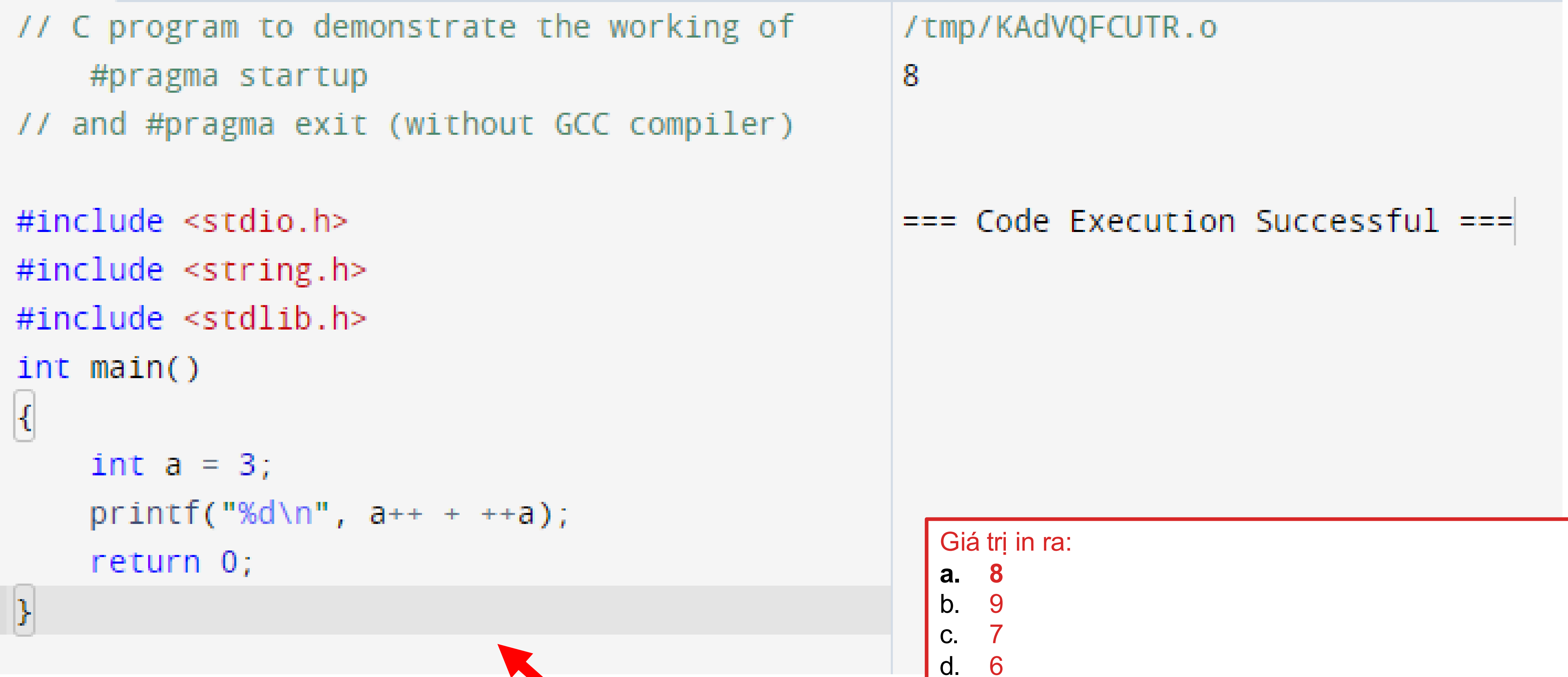
Kết quả đầu ra của chương trình:

a. **-1**

b. 1

c. 0

d. Lỗi biên dịch



```
// C program to demonstrate the working of
    #pragma startup
// and #pragma exit (without GCC compiler)

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
int main()
{
    int a = 3;
    printf("%d\n", a++ + ++a);
    return 0;
}
```

```
/tmp/KAdVQFCUTR.o
8



=== Code Execution Successful ===
```

Giá trị in ra:

a. **8**

b. 9

c. 7

d. 6

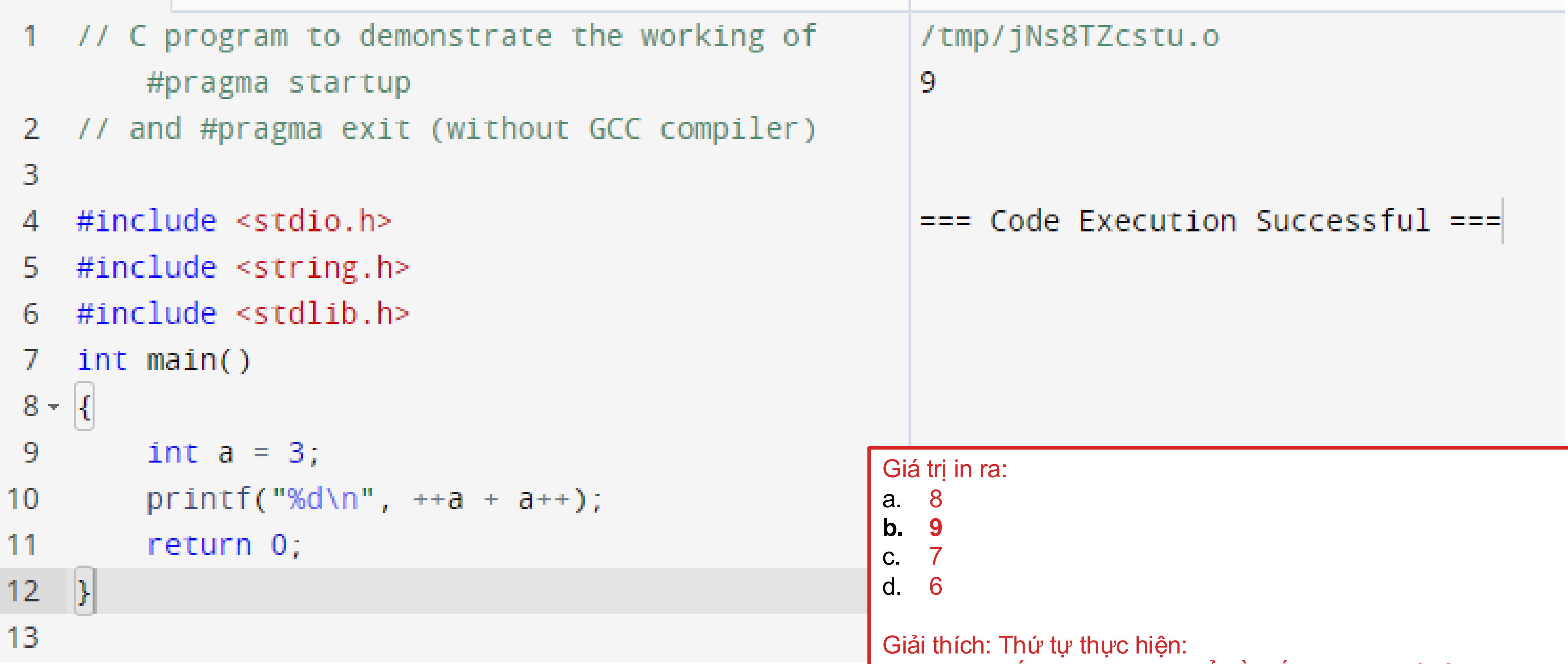
Giải thích: Thứ tự thực hiện:

a++ : tang giá trị a lên 1, sau đó trả về giá trị khi chưa tang

++a : Trực tiếp tang a lên 1

+ : cộng 2 kết quả

-> 3 + 5 = 8



```c
// C program to demonstrate the working of #pragma startup
// and #pragma exit (without GCC compiler)

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
int main()
{
    int a = 3;
    printf("%d\n", ++a + a++);
    return 0;
}

```

```
/tmp/jNs8TZcstu.o
9


=== Code Execution Successful ===
```

Giá trị in ra:

a. 8

b. 9

c. 7

d. 6

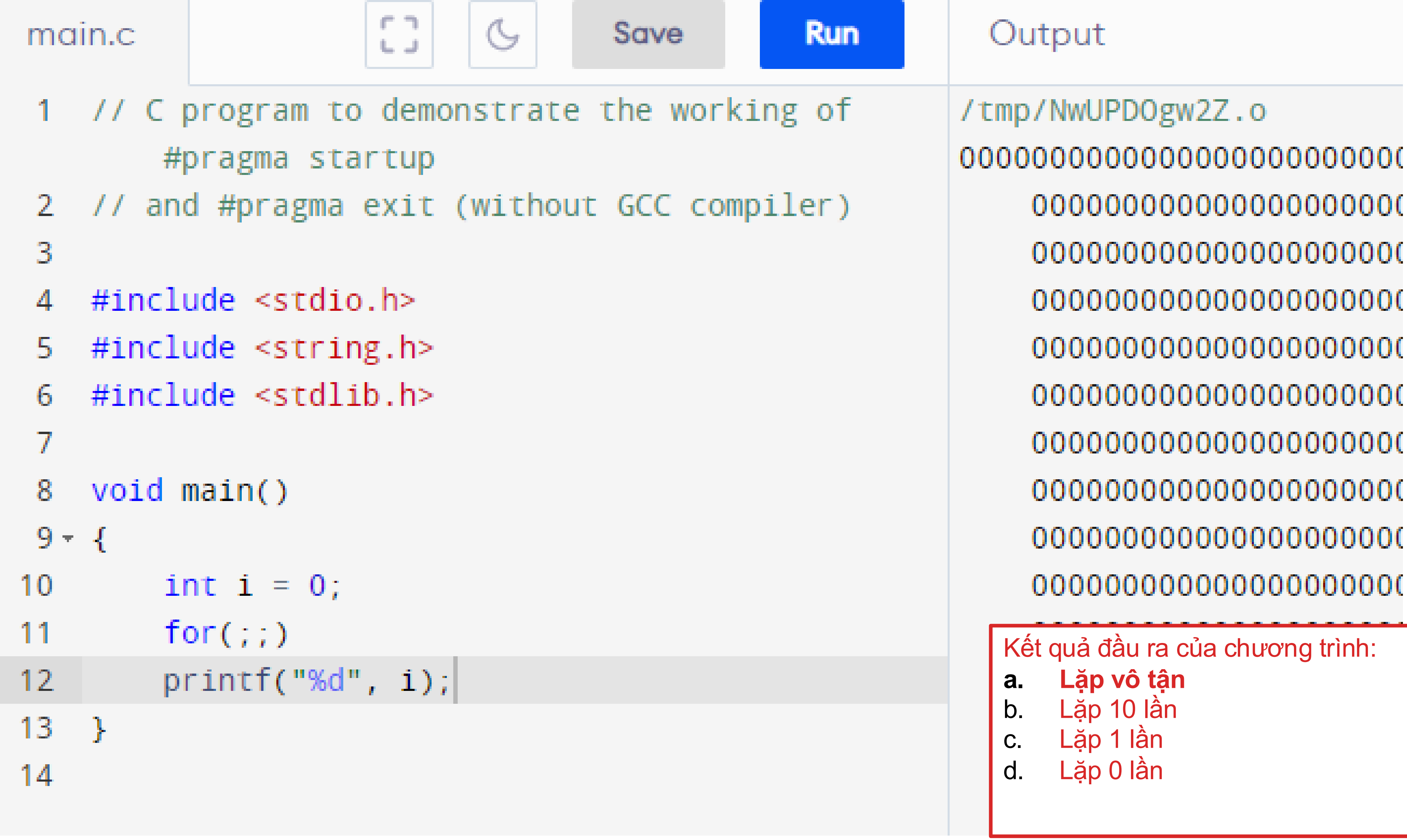
Giải thích: Thứ tự thực hiện:

++a : Trực tiếp tang a lên 1, trả về giá trị tại địa chỉ của a cho phép toán tiếp theo

a++ : tang giá trị a lên 1, sau đó trả về giá trị khi chưa tang

+ : cộng 2 kết quả (giá trị tại địa chỉ của a + giá trị a++)

-> 5 + 4 = 9



```c
// C program to demonstrate the working of
    #pragma startup
// and #pragma exit (without GCC compiler)

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

void main()
{
    int i = 0;
    for(;;)
    printf("%d", i);
}

```

```
/tmp/NwUPDOgw2Z.o
000000000000000000000000
    00000000000000000000
    00000000000000000000
    00000000000000000000
    00000000000000000000
    00000000000000000000
    00000000000000000000
    00000000000000000000
    00000000000000000000
    00000000000000000000
```

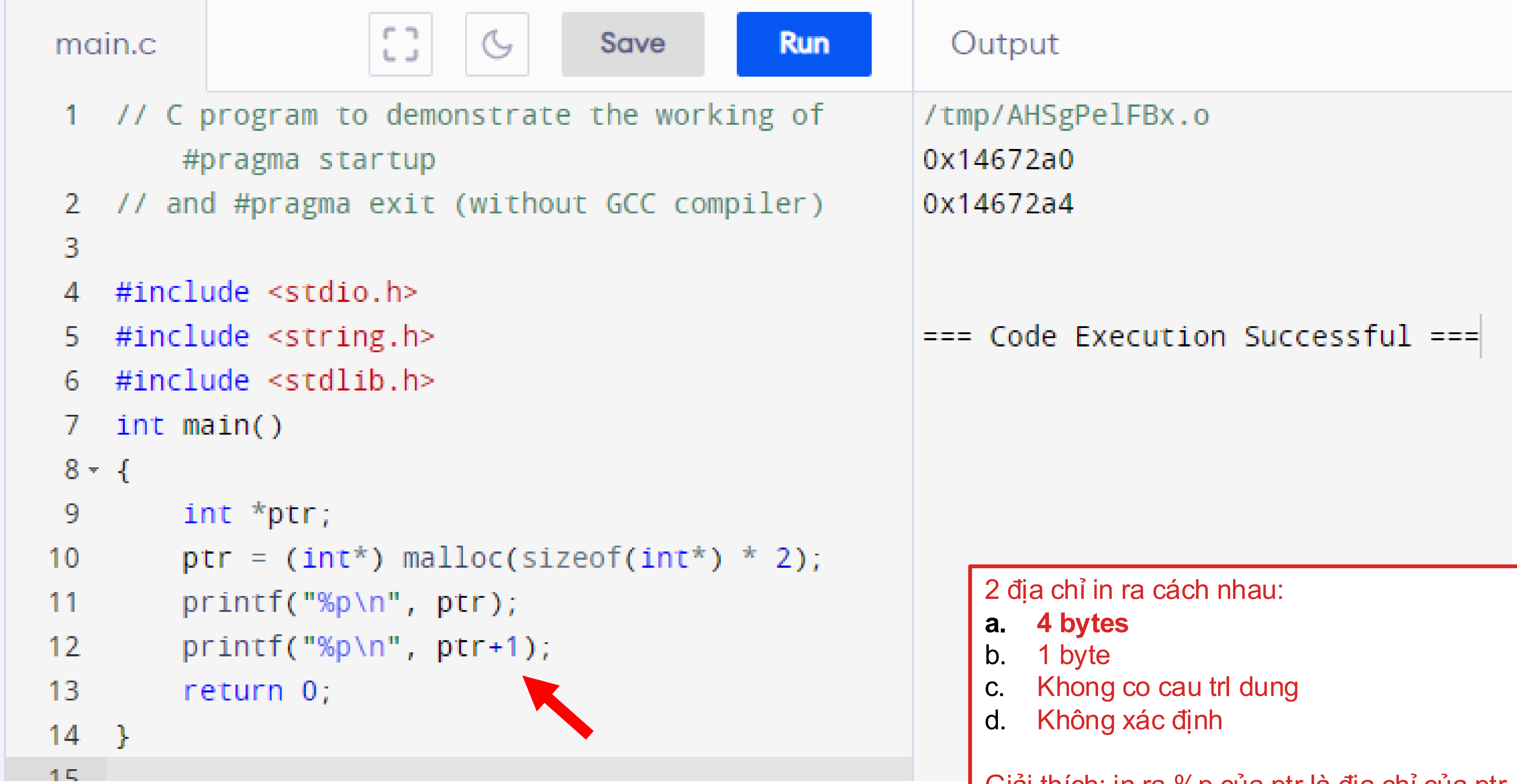
Kết quả đầu ra của chương trình:

a. **Lặp vô tận**

b. Lặp 10 lần

c. Lặp 1 lần

d. Lặp 0 lần



```c
// C program to demonstrate the working of #pragma startup
// and #pragma exit (without GCC compiler)

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
int main()
{
    int *ptr;
    ptr = (int*) malloc(sizeof(int*) * 2);
    printf("%p\n", ptr);
    printf("%p\n", ptr+1);
    return 0;
}
```

```
/tmp/AHSgPelFBx.o
0x14672a0
0x14672a4


=== Code Execution Successful ===
```

2 địa chỉ in ra cách nhau:

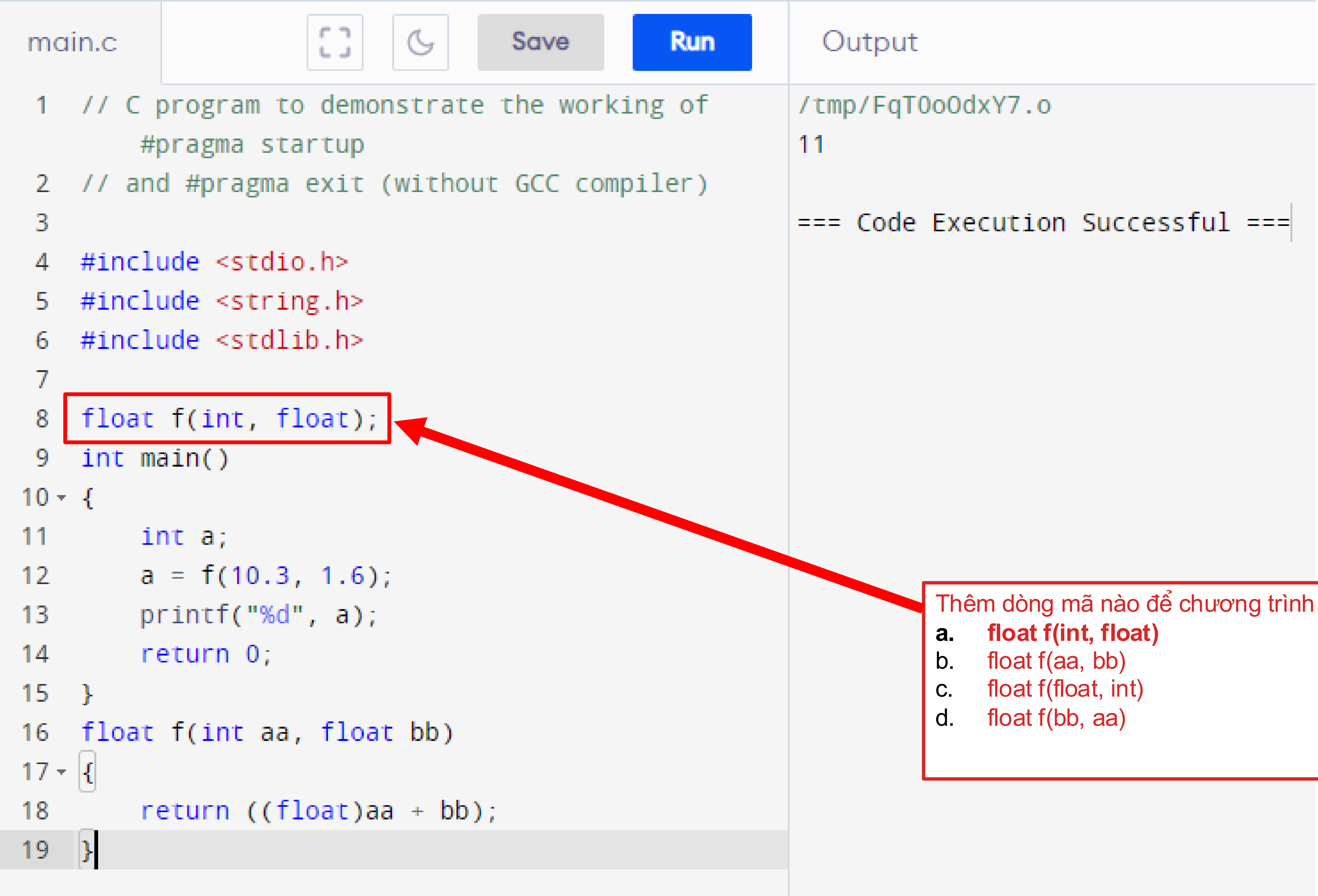
a. **4 bytes**

b. 1 byte

c. Khong co cau trl dung

d. Không xác định

Giải thích: in ra %p của ptr là địa chỉ của ptr, ptr + 1 tang địa chỉ của ptr lên 4 bytes = kích thước của int



```c
// C program to demonstrate the working of #pragma startup
// and #pragma exit (without GCC compiler)

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

float f(int, float);
int main()
{
    int a;
    a = f(10.3, 1.6);
    printf("%d", a);
    return 0;
}
float f(int aa, float bb)
{
    return ((float)aa + bb);
}
```

```
/tmp/FqT0oOdxY7.o
11


=== Code Execution Successful ===
```

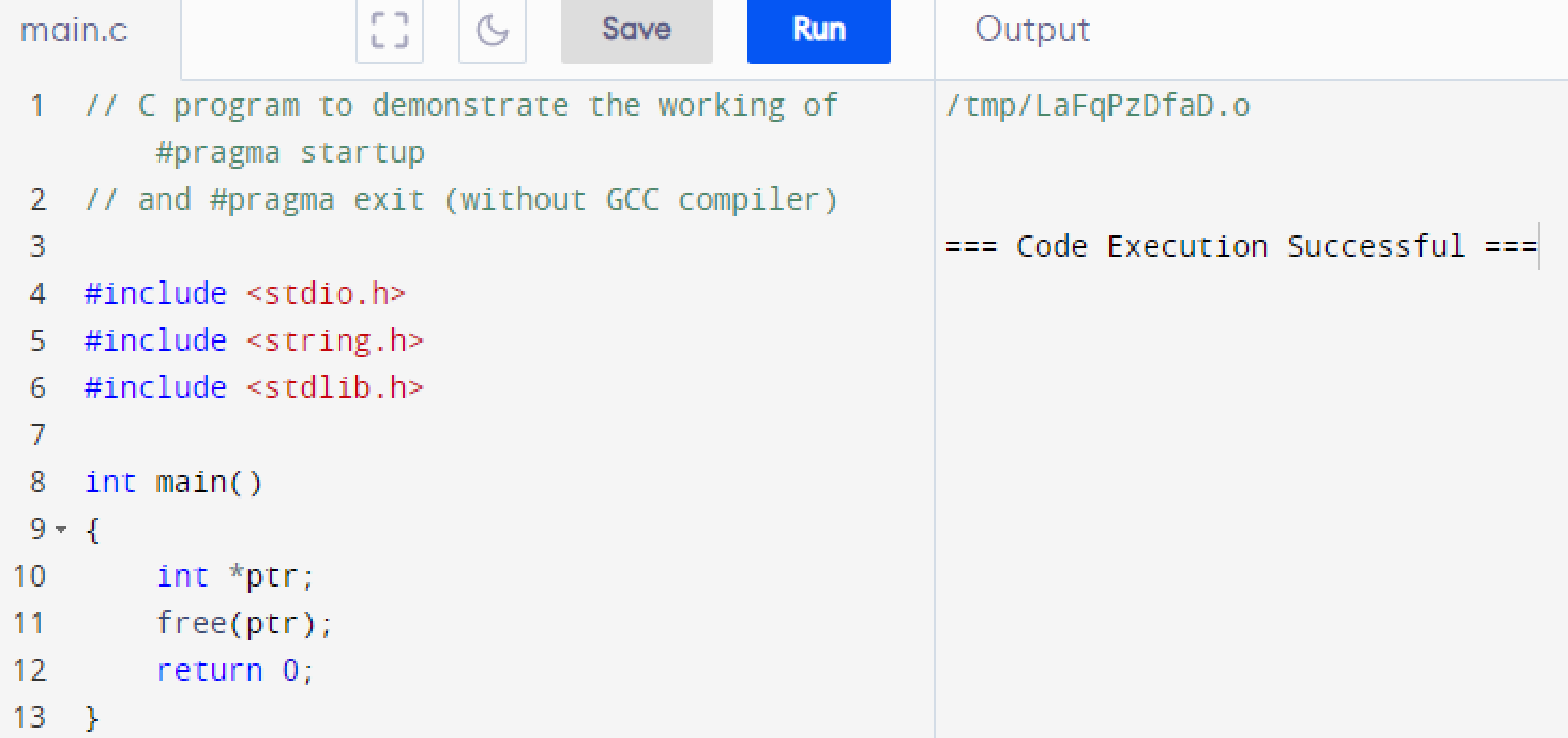
Thêm dòng mã nào để chương trình chạy đúng:

a. float f(int, float)

b. float f(aa, bb)

c. float f(float, int)

d. float f(bb, aa)



```c
// C program to demonstrate the working of #pragma startup
// and #pragma exit (without GCC compiler)

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int main()
{
    int *ptr;
    free(ptr);
    return 0;
}
```

```
/tmp/LaFqPzDfaD.o


=== Code Execution Successful ===
```

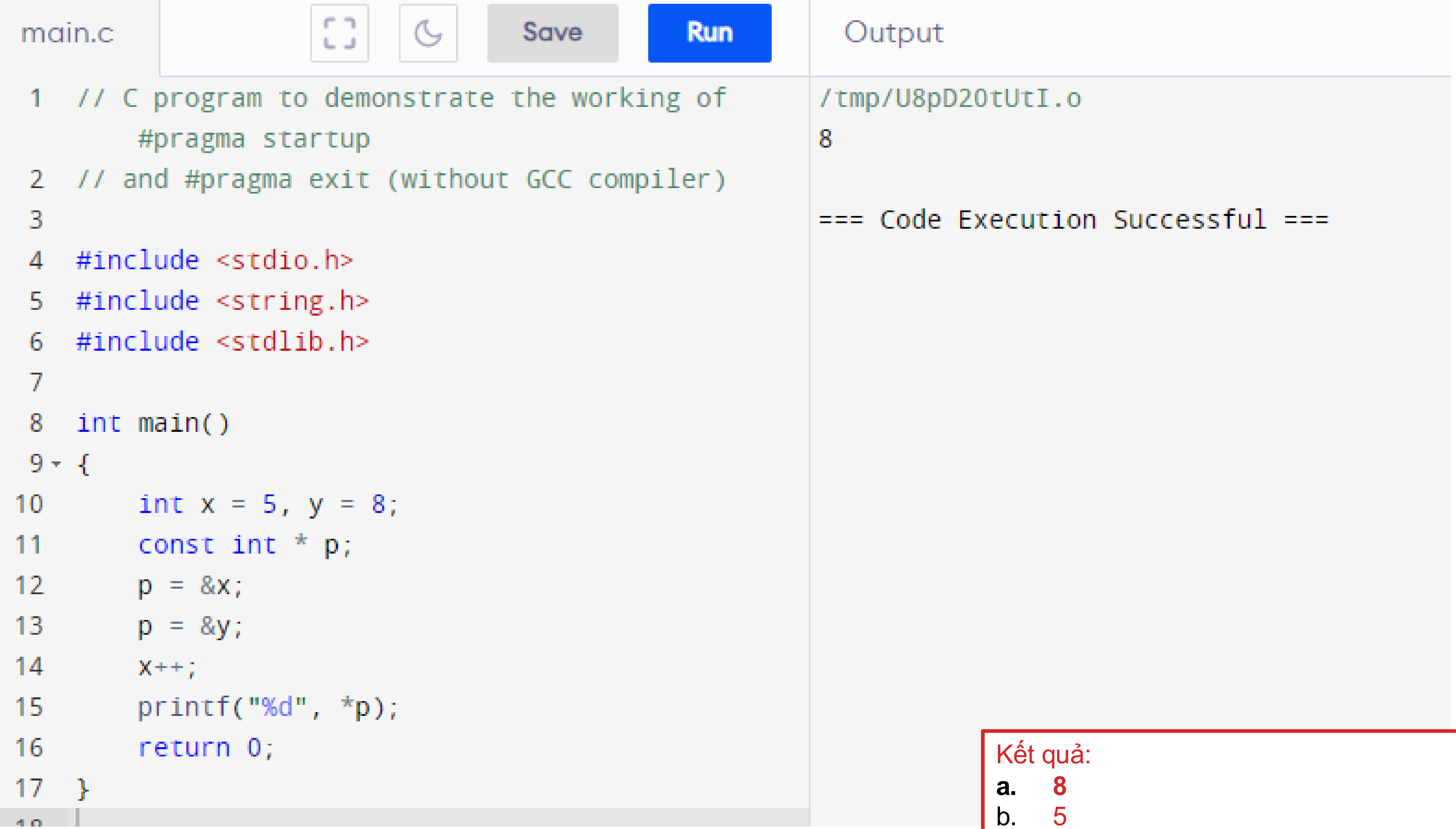
Kết quả:

a. **Không in ra gì**

b. Segment fault

c. Absort Core dumped

d. Không có câu trả lời đúng



```c
// C program to demonstrate the working of #pragma startup
// and #pragma exit (without GCC compiler)

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int main()
{
    int x = 5, y = 8;
    const int * p;
    p = &x;
    p = &y;
    x++;
    printf("%d", *p);
    return 0;
}
```

```
/tmp/U8pD2OtUtI.o
8

=== Code Execution Successful ===
```

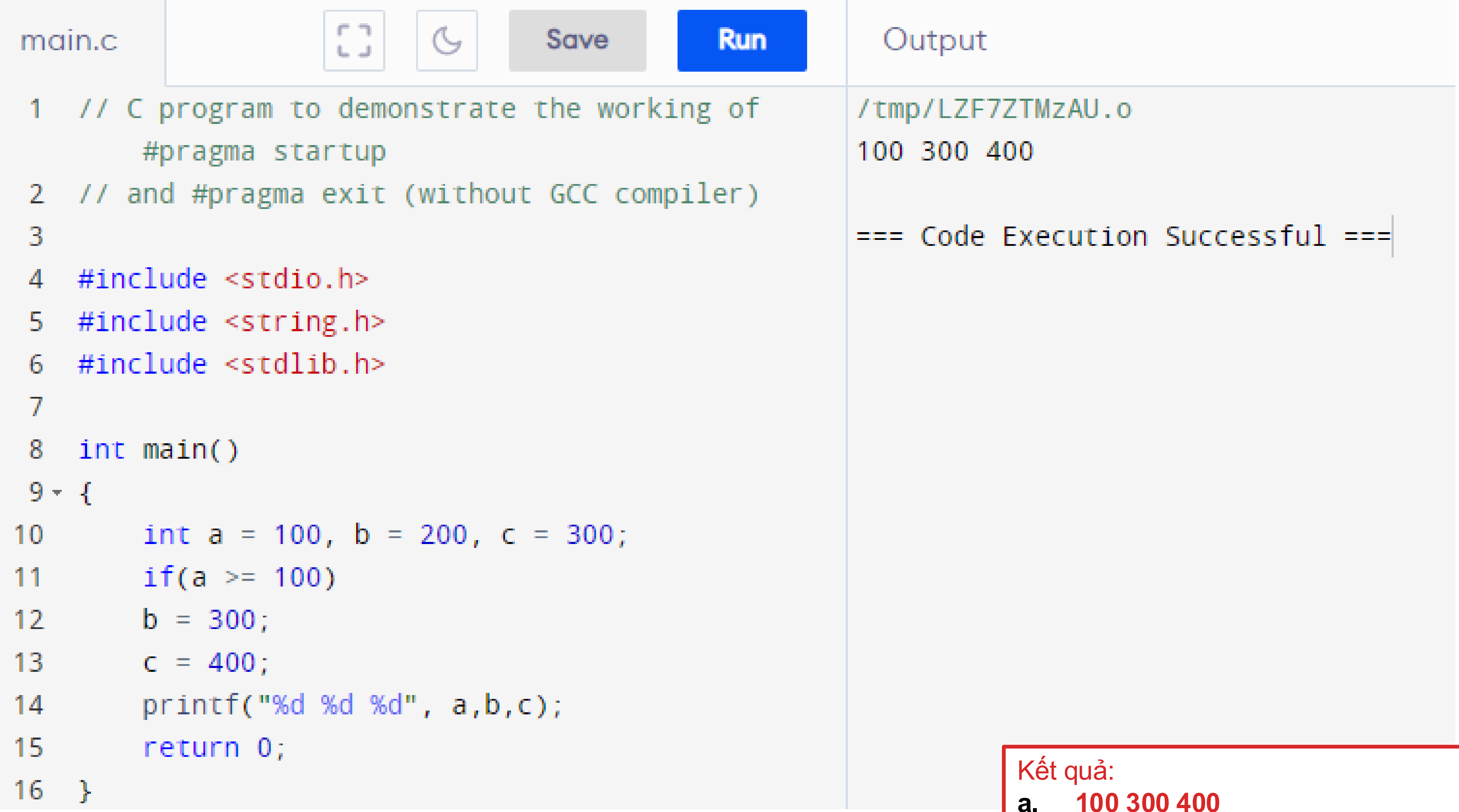
Kết quả:

a. **8**

b. 5

c. Biên dịch lỗi

d. 6



```c
// C program to demonstrate the working of #pragma startup
// and #pragma exit (without GCC compiler)

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int main()
{
    int a = 100, b = 200, c = 300;
    if(a >= 100)
    b = 300;
    c = 400;
    printf("%d %d %d", a,b,c);
    return 0;
}
```

```
/tmp/LZF7ZTMzAU.o
100 300 400

=== Code Execution Successful ===
```

Kết quả:

a. **100 300 400**

b. 100 200 300

c. 100 200 400

d. 100 300 300

Kiểu dữ liệu nào là phù hợp nhất để lưu trữ giá trị 65000 trong hệ điều hành 32 bit:

a. unsign int

b. signed int

c. int

d. long

Tùy chọn nào trong lệnh gcc sẽ bao gồm thông tin debugging trong file object:

a. -g

b. -c

c. -p

d. Không có đáp án đúng

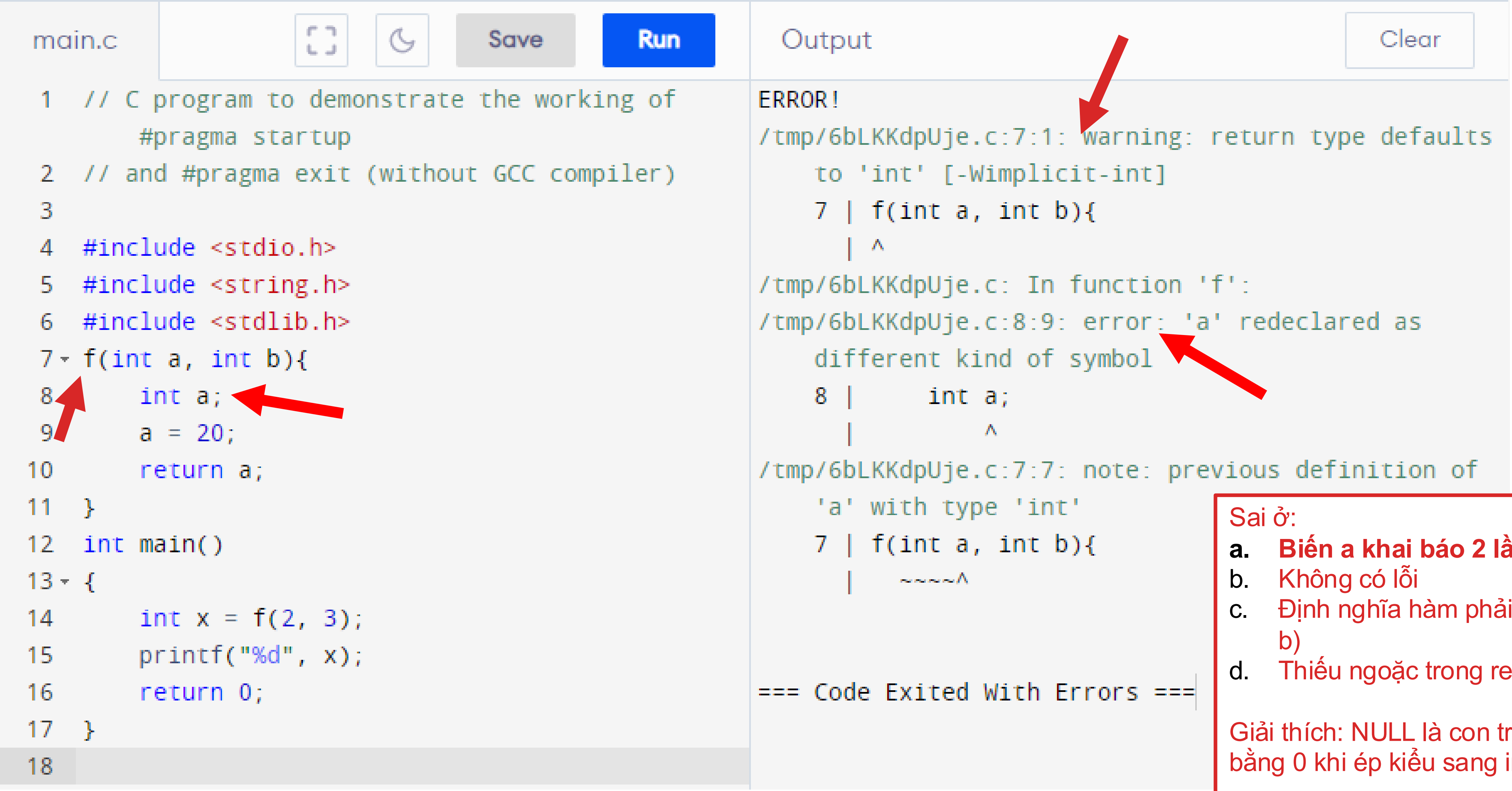
Tùy chọn nào trong lệnh gcc sẽ bao gồm thông tin debugging trong file object:

a. -g

b. -c

c. -p

d. Không có đáp án đúng



```c
// C program to demonstrate the working of #pragma startup
// and #pragma exit (without GCC compiler)

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
f(int a, int b){
    int a;
    a = 20;
    return a;
}
int main()
{
    int x = f(2, 3);
    printf("%d", x);
    return 0;
}
```

```
ERROR!
/tmp/6bLKKdpUje.c:7:1: warning: return type defaults
    to 'int' [-Wimplicit-int]
    7 | f(int a, int b){
      | ^
/tmp/6bLKKdpUje.c: In function 'f':
/tmp/6bLKKdpUje.c:8:9: error: 'a' redeclared as
    different kind of symbol
    8 |     int a;
      |         ^
/tmp/6bLKKdpUje.c:7:7: note: previous definition of
    'a' with type 'int'
    7 | f(int a, int b){
      |   ~~~~^


=== Code Exited With Errors ===
```

Sai ở:

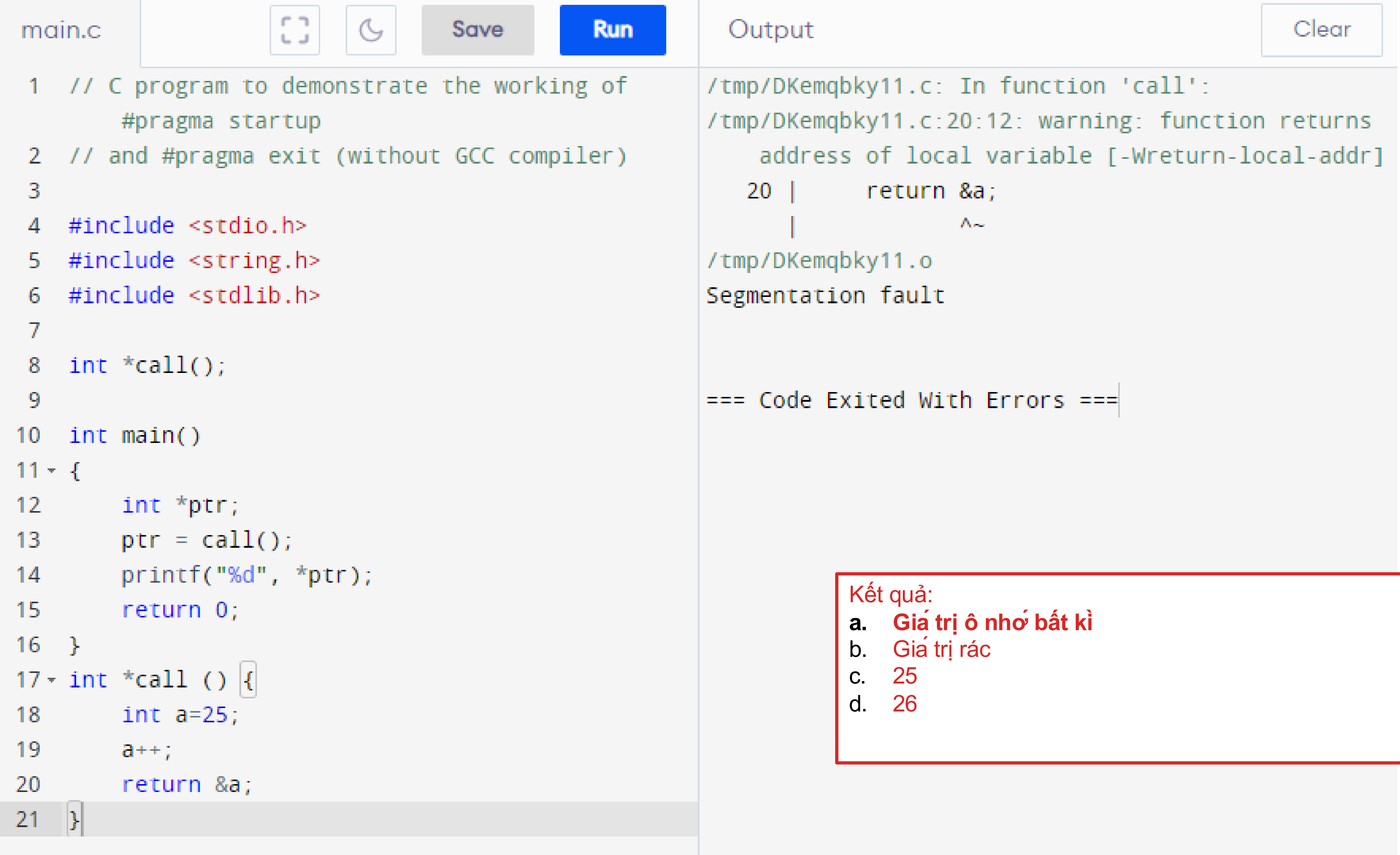
a. **Biến a khai báo 2 lần**

b. Không có lỗi

c. Định nghĩa hàm phải là int f(int a, int b)

d. Thiếu ngoặc trong return

Giải thích: NULL là con trỏ void, có giá trị bằng 0 khi ép kiểu sang int



```c
// C program to demonstrate the working of #pragma startup
// and #pragma exit (without GCC compiler)

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int *call();

int main()
{
    int *ptr;
    ptr = call();
    printf("%d", *ptr);
    return 0;
}
int *call () {
    int a=25;
    a++;
    return &a;
}
```

```
/tmp/DKemqbky11.c: In function 'call':
/tmp/DKemqbky11.c:20:12: warning: function returns
    address of local variable [-Wreturn-local-addr]
   20 |     return &a;
      |            ^~
/tmp/DKemqbky11.o
Segmentation fault


=== Code Exited With Errors ===
```

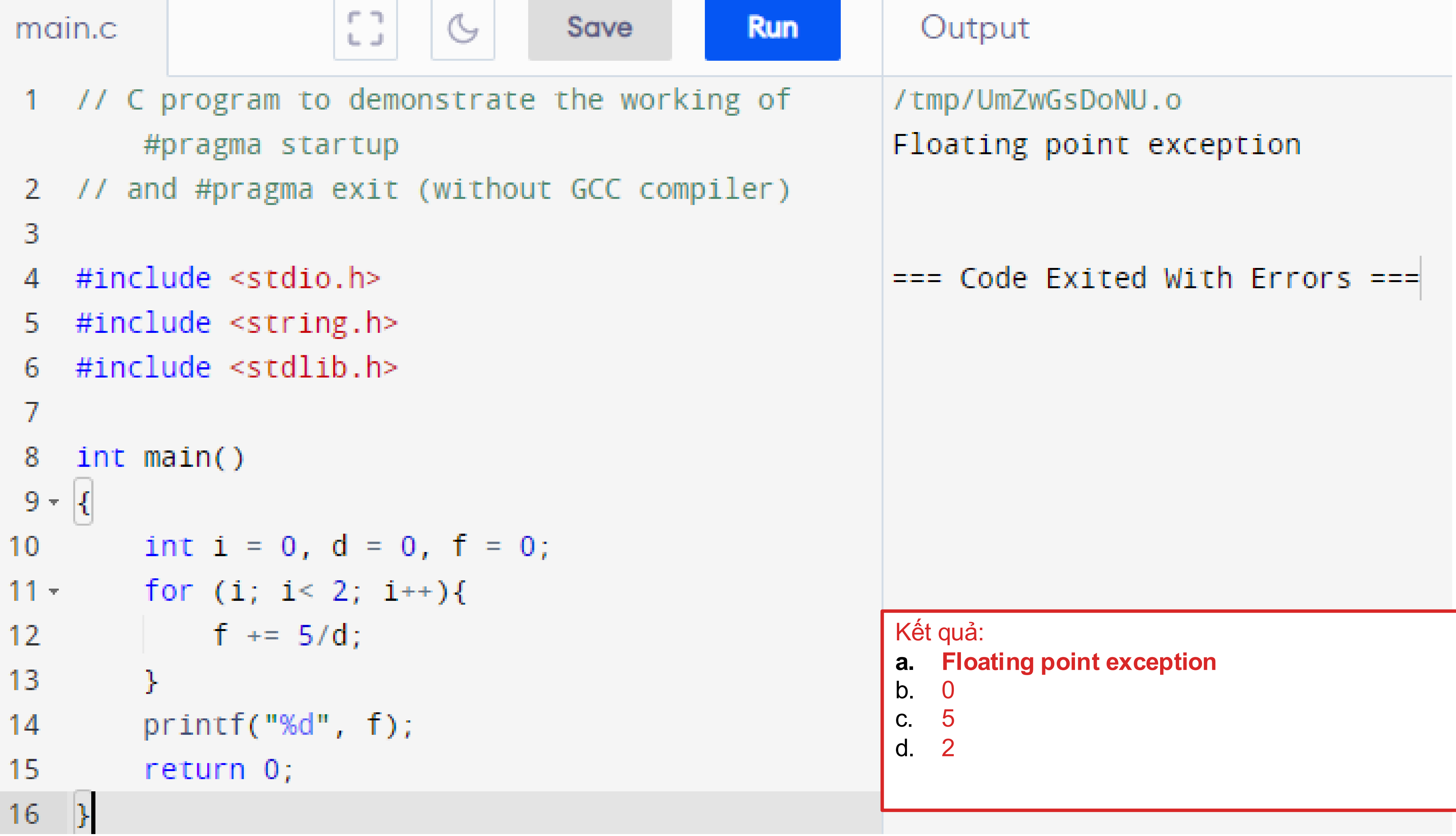
Kết quả:

a. **Giá trị ô nhớ bất kì**

b. Giá trị rác

c. 25

d. 26

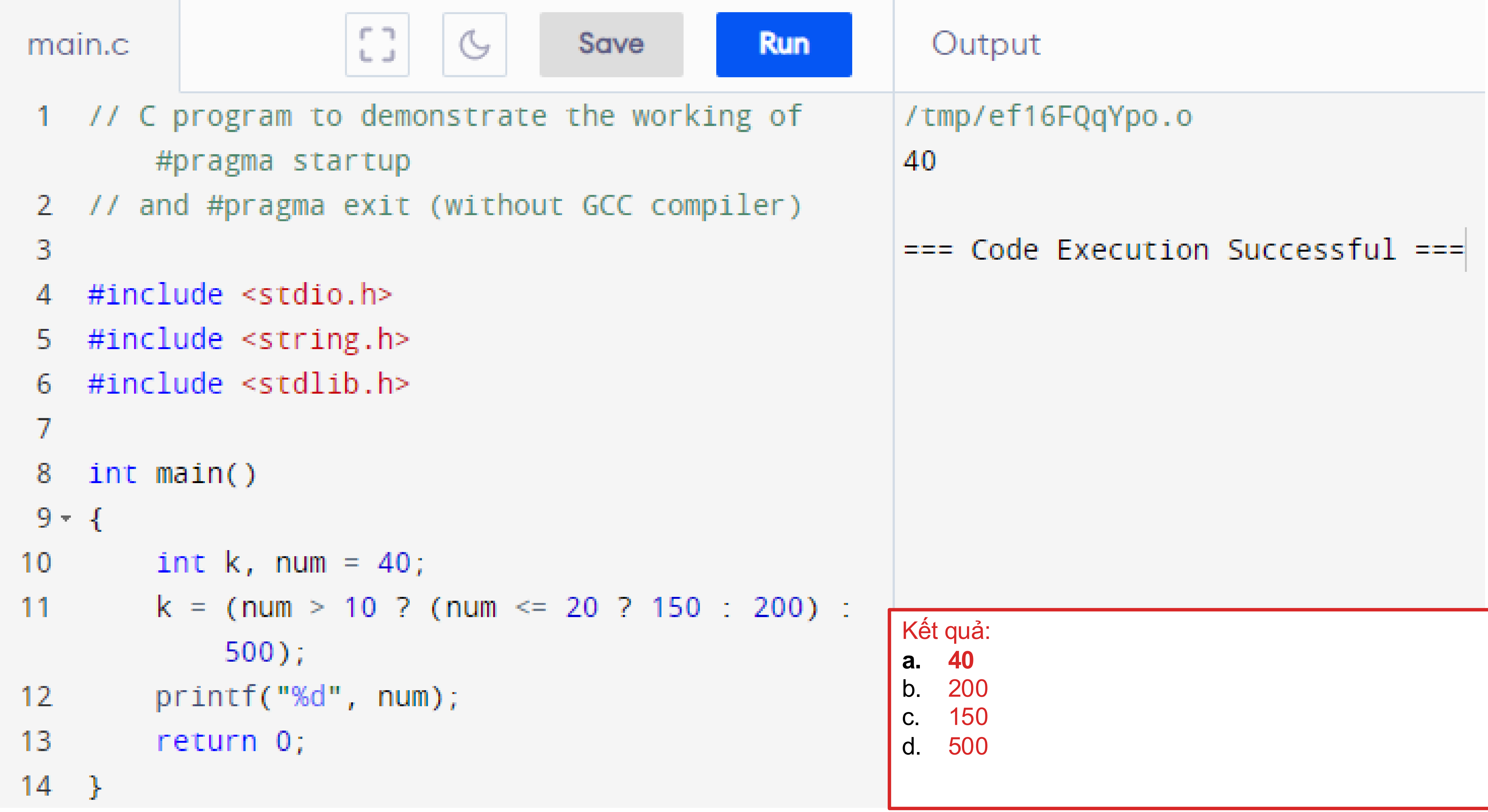


```c
// C program to demonstrate the working of #pragma startup
// and #pragma exit (without GCC compiler)

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int main()
{
    int i = 0, d = 0, f = 0;
    for (i; i< 2; i++){
        f += 5/d;
    }
    printf("%d", f);
    return 0;
}
```

```
/tmp/UmZwGsDoNU.o
Floating point exception


=== Code Exited With Errors ===
```

Kết quả:

- **a.** **Floating point exception**
- b. 0
- c. 5
- d. 2



```c
// C program to demonstrate the working of #pragma startup
// and #pragma exit (without GCC compiler)

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int main()
{
    int k, num = 40;
    k = (num > 10 ? (num <= 20 ? 150 : 200) : 500);
    printf("%d", num);
    return 0;
}
```

```
/tmp/ef16FQqYpo.o
40


=== Code Execution Successful ===
```

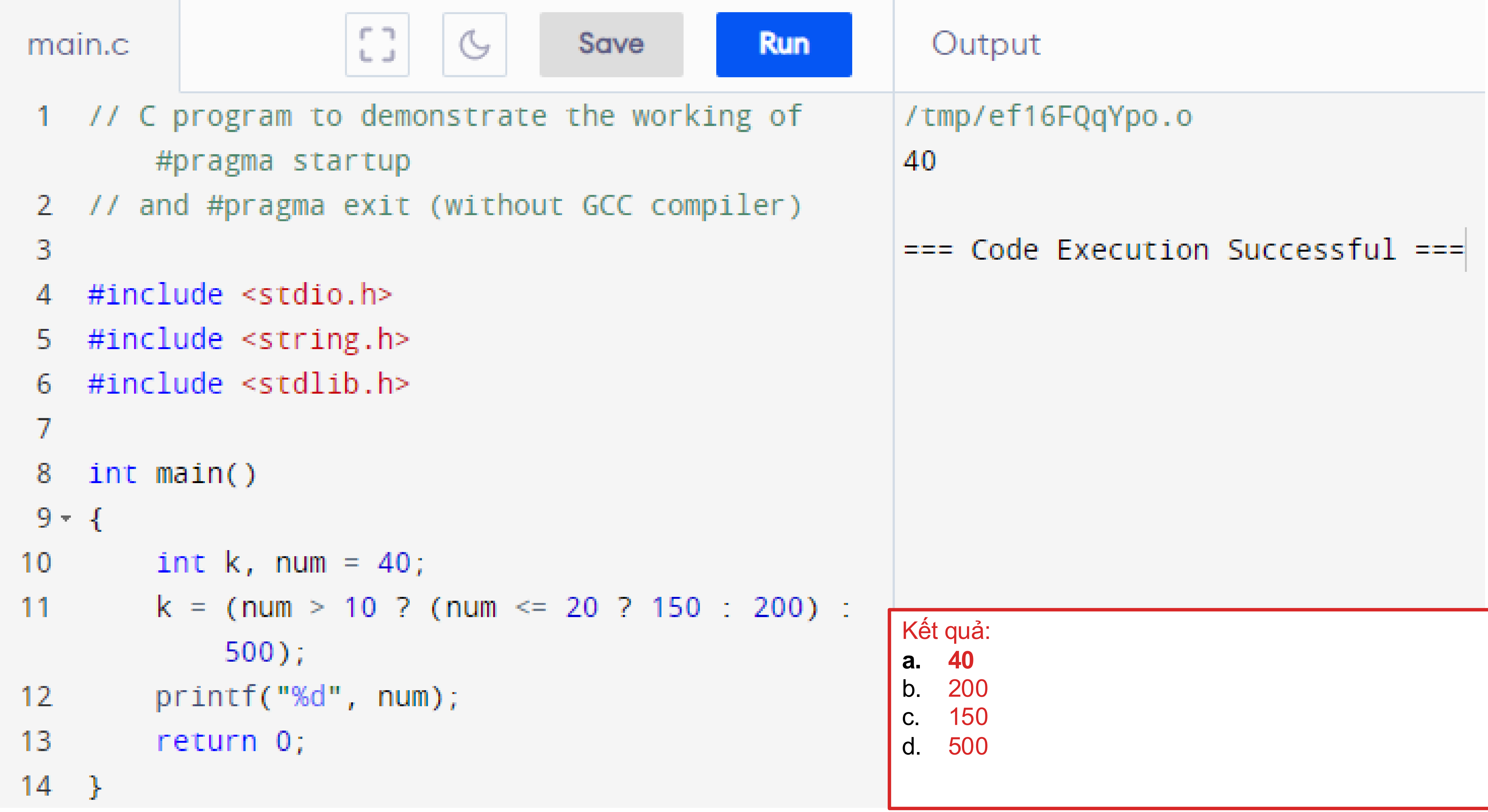
Kết quả:

a. **40**

b. 200

c. 150

d. 500



```c
// C program to demonstrate the working of #pragma startup
// and #pragma exit (without GCC compiler)

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int main()
{
    int k, num = 40;
    k = (num > 10 ? (num <= 20 ? 150 : 200) : 500);
    printf("%d", num);
    return 0;
}
```

```
/tmp/ef16FQqYpo.o
40


=== Code Execution Successful ===
```

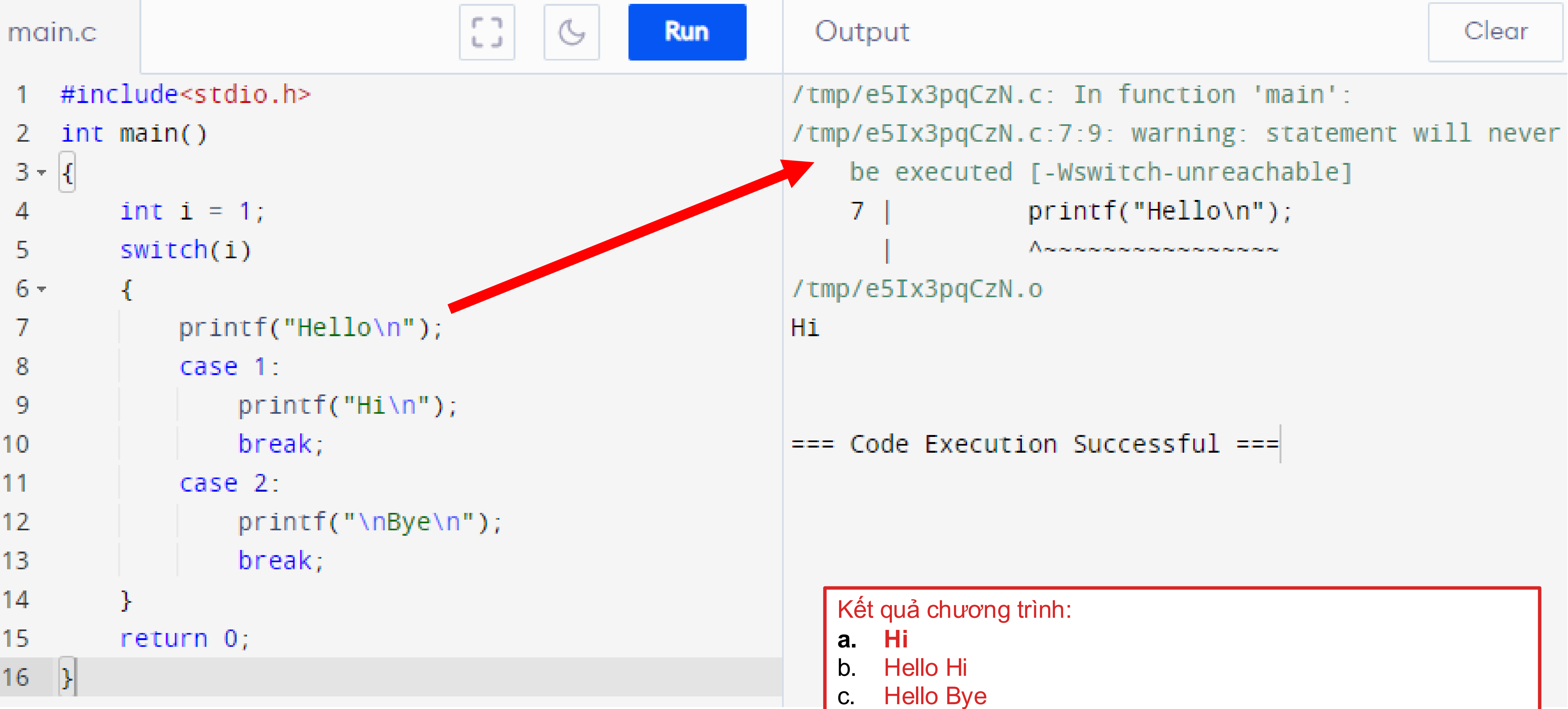
Kết quả:

a. **40**

b. 200

c. 150

d. 500



```
1  #include<stdio.h>
2  int main()
3  {
4      int i = 1;
5      switch(i)
6      {
7          printf("Hello\n");
8          case 1:
9              printf("Hi\n");
10             break;
11         case 2:
12             printf("\nBye\n");
13             break;
14     }
15     return 0;
16 }
```

```
/tmp/e5Ix3pqCzN.c: In function 'main':
/tmp/e5Ix3pqCzN.c:7:9: warning: statement will never
    be executed [-Wswitch-unreachable]
    7 |         printf("Hello\n");
      |         ^~~~~~~~~~~~~~~~~
/tmp/e5Ix3pqCzN.o
Hi


=== Code Execution Successful ===
```

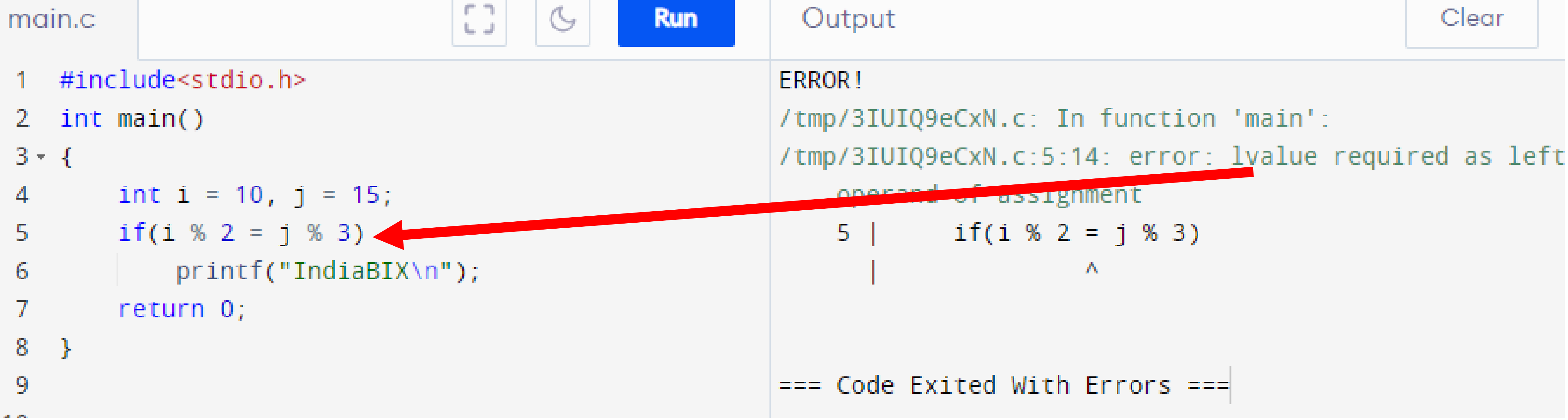
Kết quả chương trình:

a. **Hi**

b. Hello Hi

c. Hello Bye

d. Bye



```c
#include<stdio.h>
int main()
{
    int i = 10, j = 15;
    if(i % 2 = j % 3)
        printf("IndiaBIX\n");
    return 0;
}
```

```
ERROR!
/tmp/3IUIQ9eCxN.c: In function 'main':
/tmp/3IUIQ9eCxN.c:5:14: error: lvalue required as left
    operand of assignment
    5 |     if(i % 2 = j % 3)
      |              ^


=== Code Exited With Errors ===
```

Kết quả chương trình:

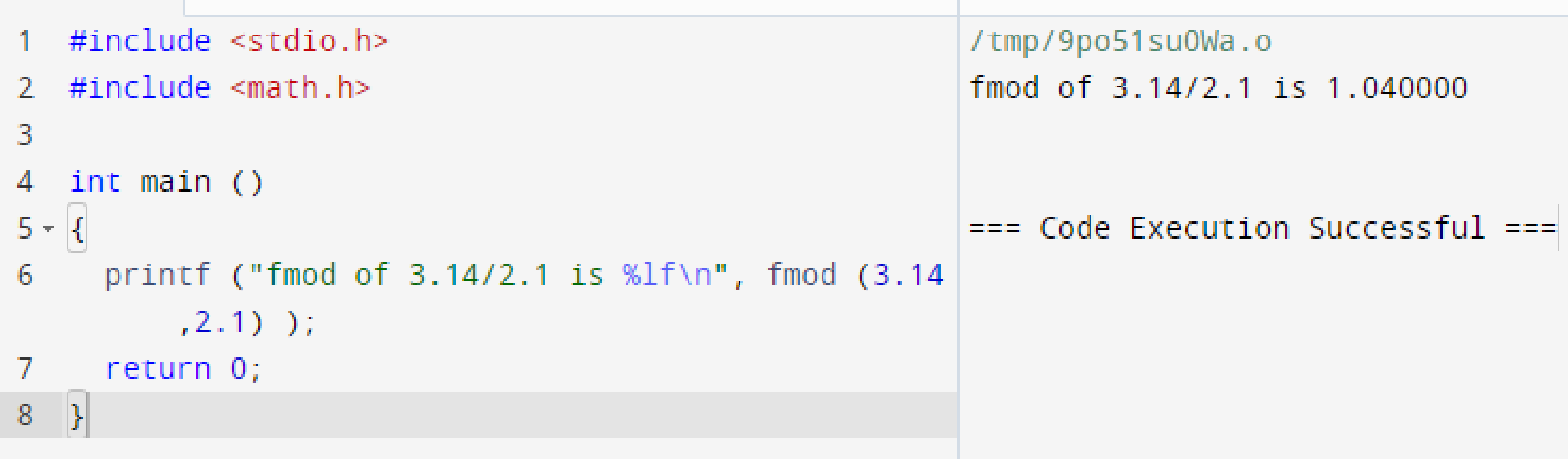
a. **Error: Lvalue required**
b. Error: Expression syntax
c. Error: Rvalue required
d. The Code runs successfully

Quá trình Preprocessor có thể phát hiện các lỗi đơn giản không?

a. Không

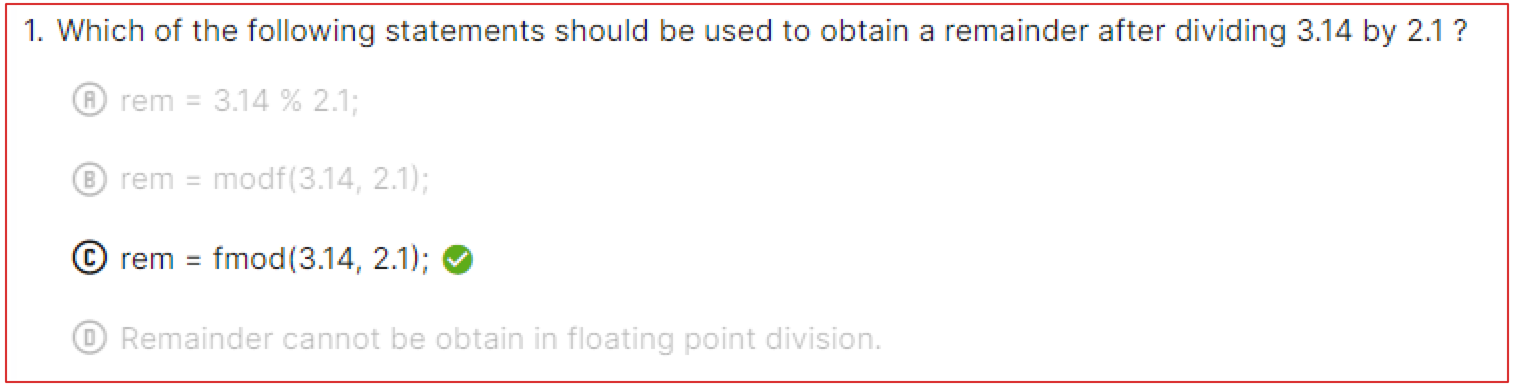
b. Có

Giải thích: Preprocessor không thể phát hiện lỗi, nó chỉ thay thế các marco với giá trị được đưa vào. Compiler sẽ phát hiện lỗi.



```c
#include <stdio.h>
#include <math.h>

int main ()
{
  printf ("fmod of 3.14/2.1 is %lf\n", fmod (3.14
      ,2.1) );
  return 0;
}
```

```
/tmp/9po51su0Wa.o
fmod of 3.14/2.1 is 1.040000


=== Code Execution Successful ===
```



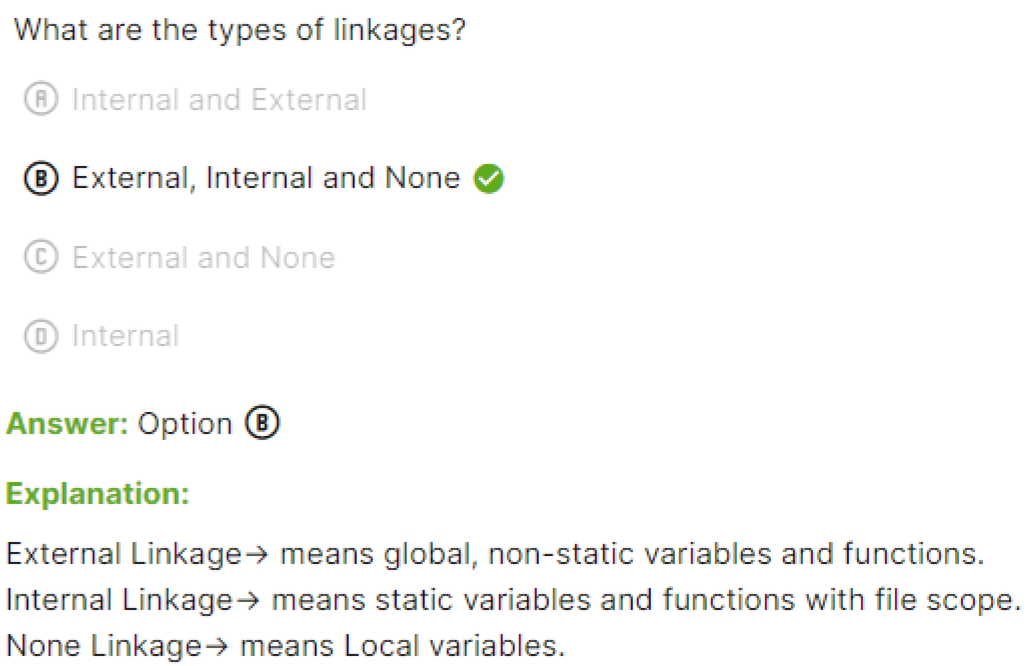
1. Which of the following statements should be used to obtain a remainder after dividing 3.14 by 2.1 ?

Ⓐ rem = 3.14 % 2.1;

Ⓑ rem = modf(3.14, 2.1);

[✓ Ⓒ rem = fmod(3.14, 2.1);] ✓

Ⓓ Remainder cannot be obtain in floating point division.



What are the types of linkages?

Ⓐ Internal and External

Ⓑ External, Internal and None ✓

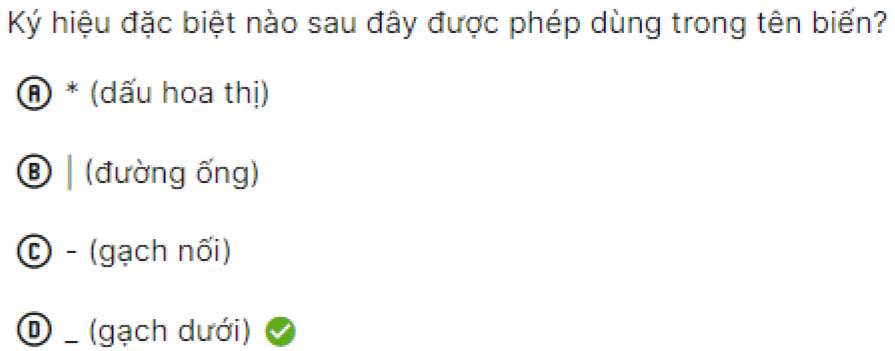
Ⓒ External and None

Ⓓ Internal

Answer: Option Ⓑ

Explanation:

External Linkage→ means global, non-static variables and functions.
Internal Linkage→ means static variables and functions with file scope.
None Linkage→ means Local variables.



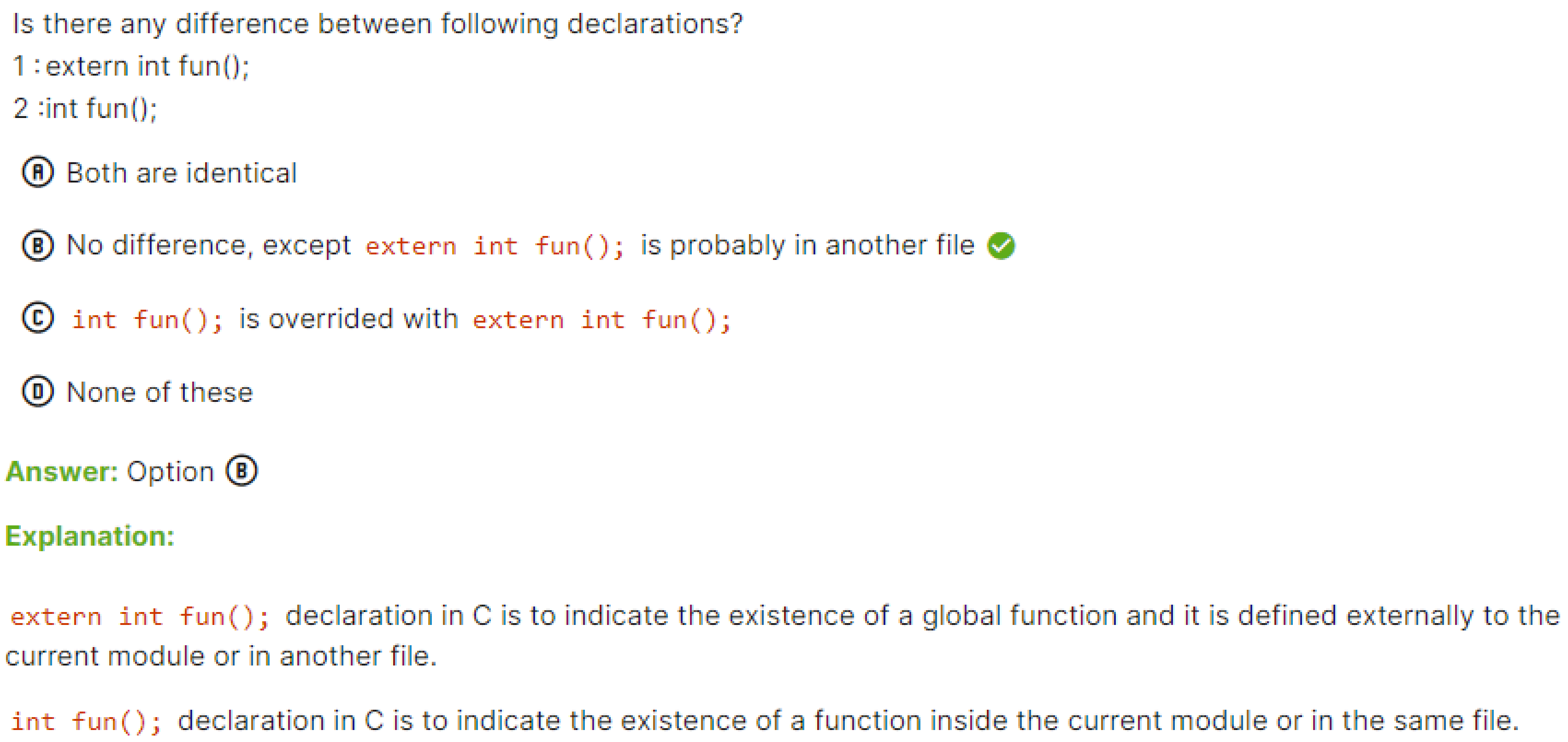
Ký hiệu đặc biệt nào sau đây được phép dùng trong tên biến?

Ⓐ * (dấu hoa thị)

Ⓑ | (đường ống)

Ⓒ - (gạch nối)

Ⓓ _ (gạch dưới) ✓



Is there any difference between following declarations?
1 : extern int fun();
2 :int fun();

Ⓐ Both are identical

Ⓑ No difference, except `extern int fun();` is probably in another file ✓

Ⓒ `int fun();` is overrided with `extern int fun();`

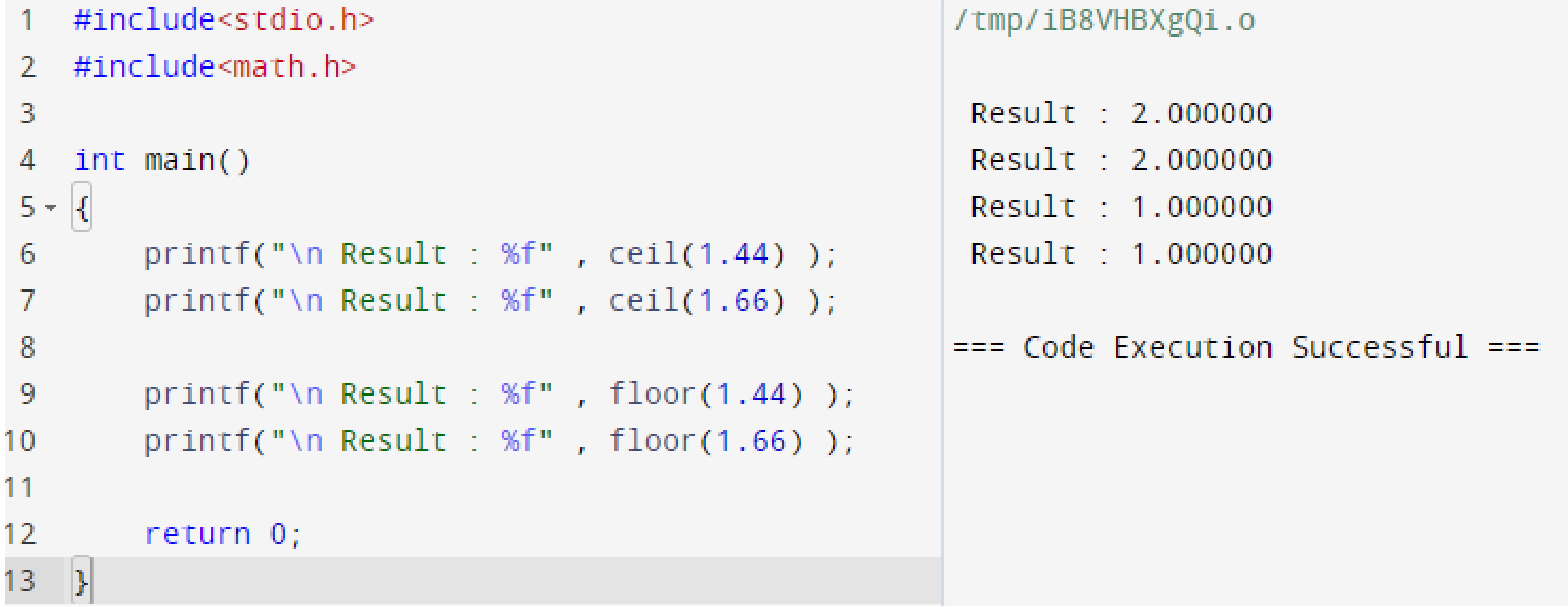
Ⓓ None of these

Answer: Option Ⓑ

Explanation:

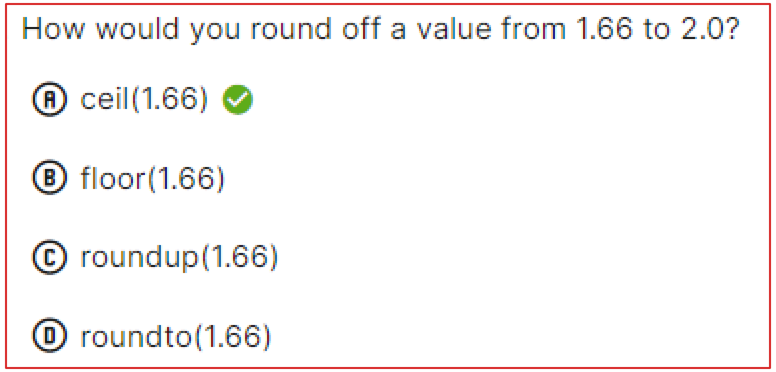
`extern int fun();` declaration in C is to indicate the existence of a global function and it is defined externally to the current module or in another file.

`int fun();` declaration in C is to indicate the existence of a function inside the current module or in the same file.



```c
#include<stdio.h>
#include<math.h>

int main()
{
    printf("\n Result : %f" , ceil(1.44) );
    printf("\n Result : %f" , ceil(1.66) );

    printf("\n Result : %f" , floor(1.44) );
    printf("\n Result : %f" , floor(1.66) );

    return 0;
}
```

```
/tmp/iB8VHBXgQi.o

 Result : 2.000000
 Result : 2.000000
 Result : 1.000000
 Result : 1.000000

=== Code Execution Successful ===
```



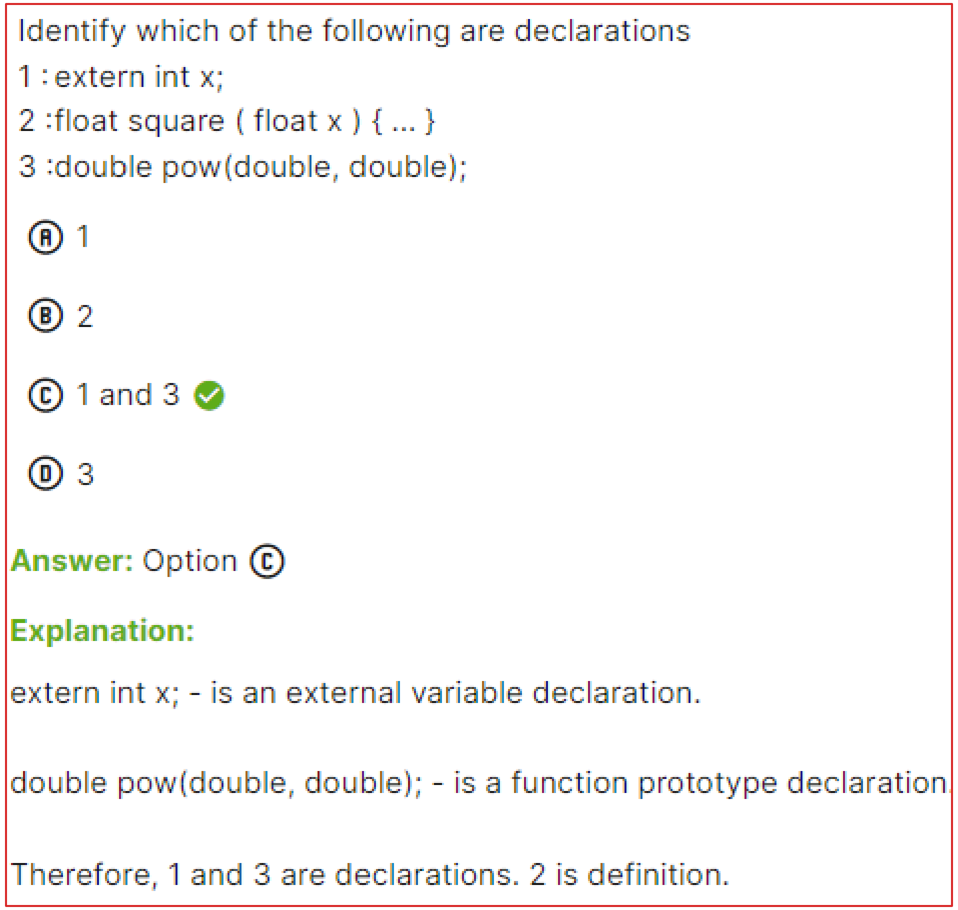
How would you round off a value from 1.66 to 2.0?

Ⓐ ceil(1.66) ✓

Ⓑ floor(1.66)

Ⓒ roundup(1.66)

Ⓓ roundto(1.66)



Identify which of the following are declarations

1 : extern int x;

2 :float square (float x) { ... }

3 :double pow(double, double);

Ⓐ 1

Ⓑ 2

Ⓒ 1 and 3 ✓

Ⓓ 3

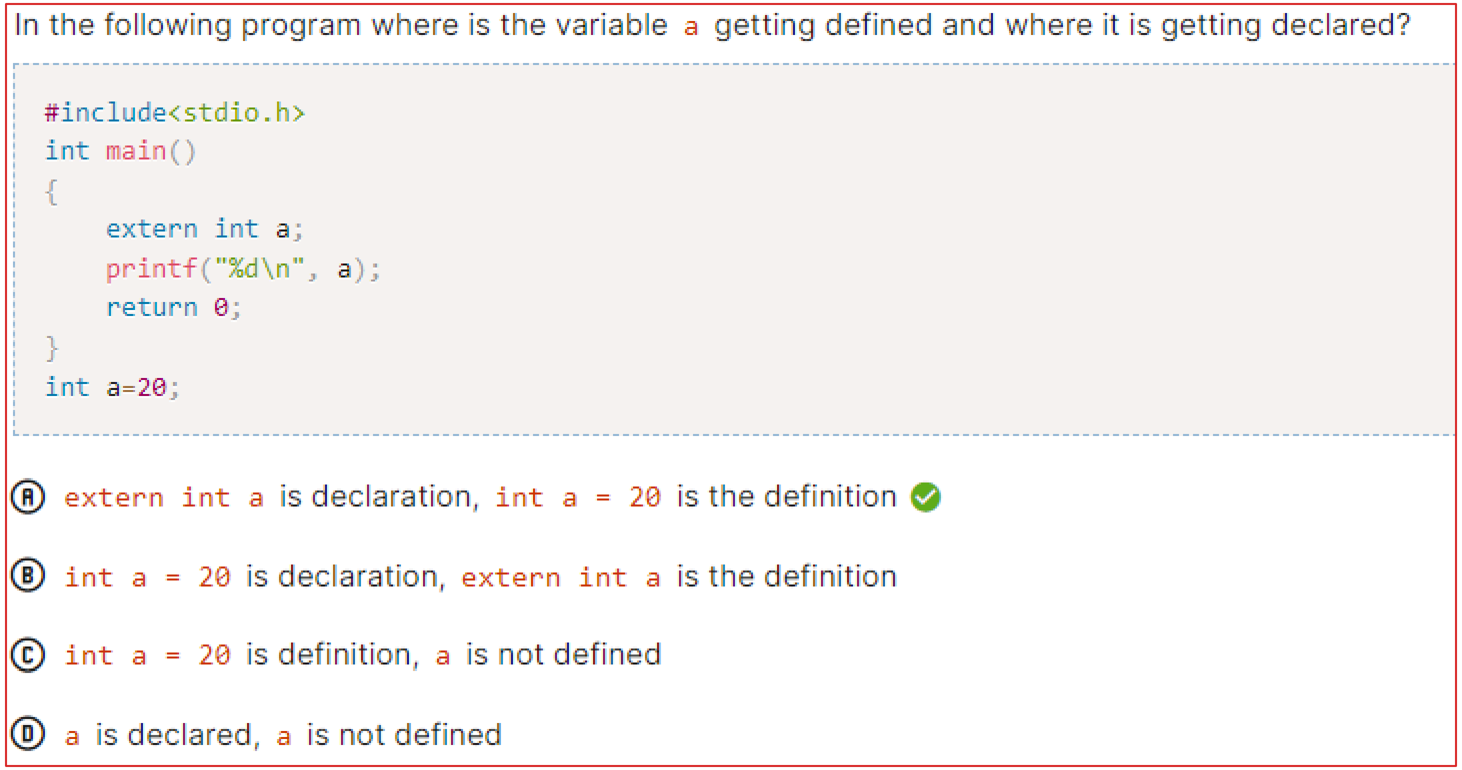
Answer: Option Ⓒ

Explanation:

extern int x; - is an external variable declaration.

double pow(double, double); - is a function prototype declaration

Therefore, 1 and 3 are declarations. 2 is definition.



In the following program where is the variable a getting defined and where it is getting declared?

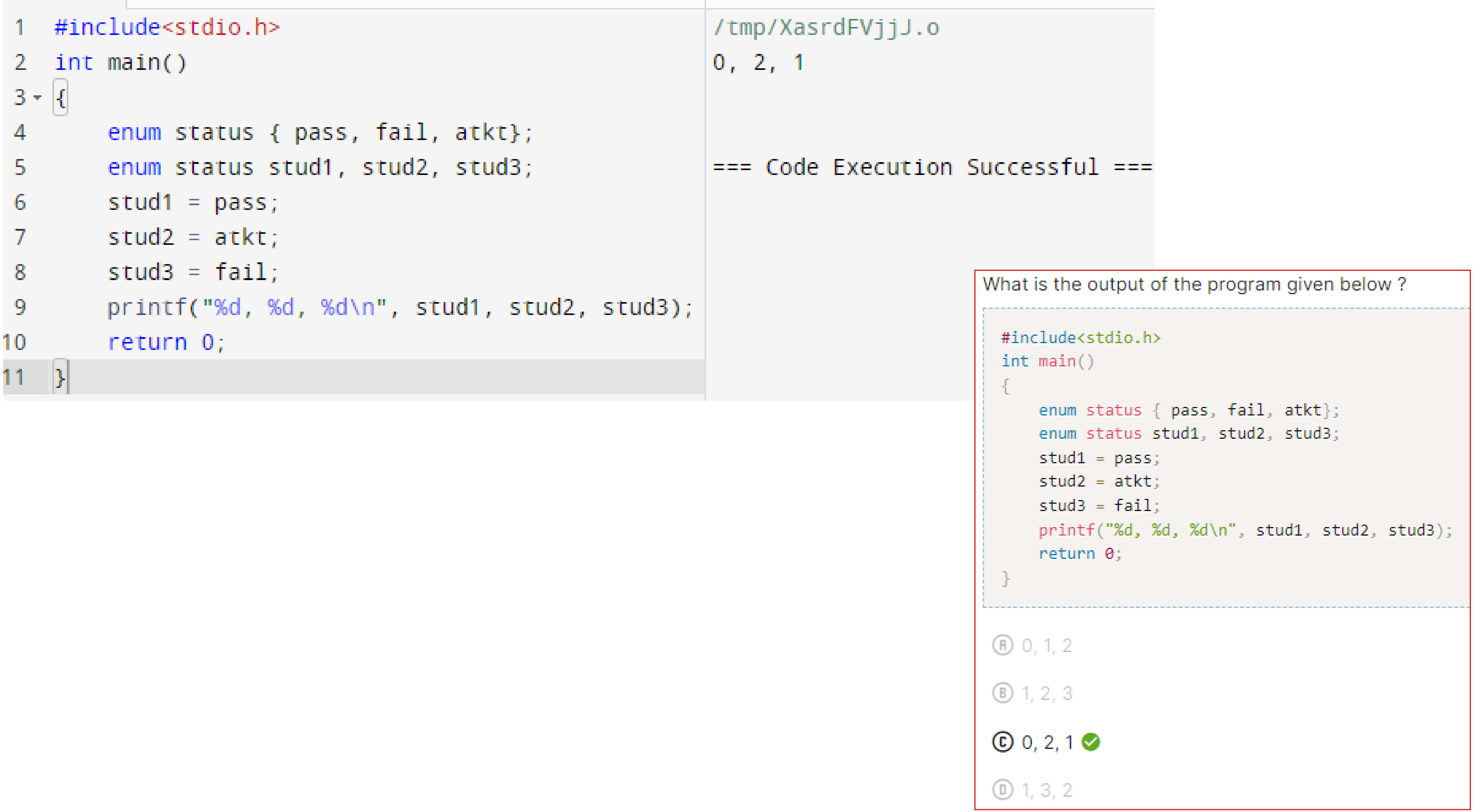
```
#include<stdio.h>
int main()
{
    extern int a;
    printf("%d\n", a);
    return 0;
}
int a=20;
```

Ⓐ `extern int a` is declaration, `int a = 20` is the definition ✓

Ⓑ `int a = 20` is declaration, `extern int a` is the definition

Ⓒ `int a = 20` is definition, `a` is not defined

Ⓓ `a` is declared, `a` is not defined



```
#include<stdio.h>
int main()
{
    enum status { pass, fail, atkt};
    enum status stud1, stud2, stud3;
    stud1 = pass;
    stud2 = atkt;
    stud3 = fail;
    printf("%d, %d, %d\n", stud1, stud2, stud3);
    return 0;
}
```

```
/tmp/XasrdFVjjJ.o
0, 2, 1


=== Code Execution Successful ===
```

What is the output of the program given below ?

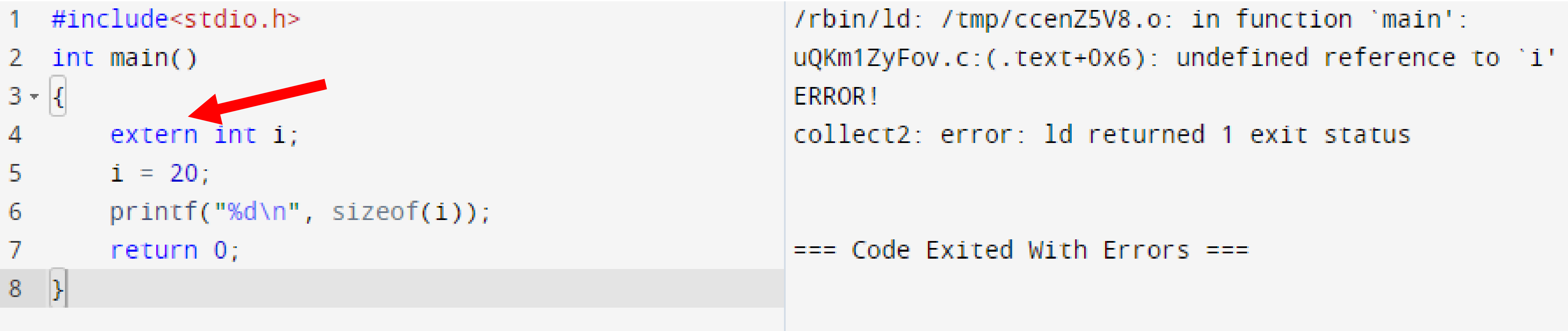
```
#include<stdio.h>
int main()
{
    enum status { pass, fail, atkt};
    enum status stud1, stud2, stud3;
    stud1 = pass;
    stud2 = atkt;
    stud3 = fail;
    printf("%d, %d, %d\n", stud1, stud2, stud3);
    return 0;
}
```

Ⓐ 0, 1, 2

Ⓑ 1, 2, 3

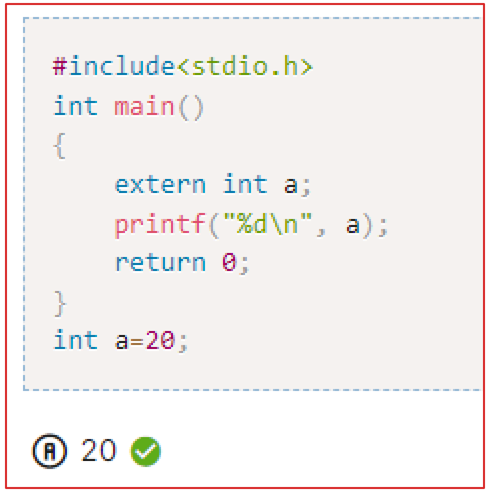
Ⓒ 0, 2, 1 ✓

Ⓓ 1, 3, 2



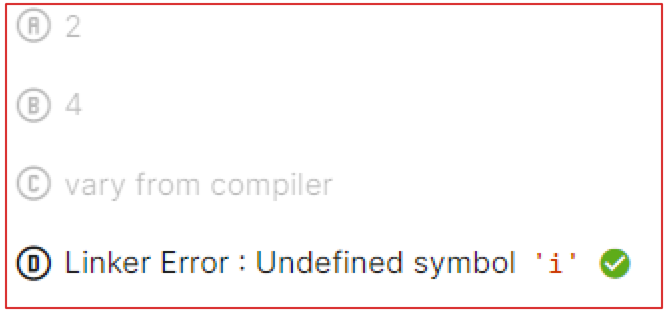
```c
#include<stdio.h>
int main()
{
    extern int i;
    i = 20;
    printf("%d\n", sizeof(i));
    return 0;
}
```

```
/rbin/ld: /tmp/ccenZ5V8.o: in function `main':
uQKm1ZyFov.c:(.text+0x6): undefined reference to `i'
ERROR!
collect2: error: ld returned 1 exit status


=== Code Exited With Errors ===
```



```c
#include<stdio.h>
int main()
{
    extern int a;
    printf("%d\n", a);
    return 0;
}
int a=20;
```

Ⓐ 20 ✓

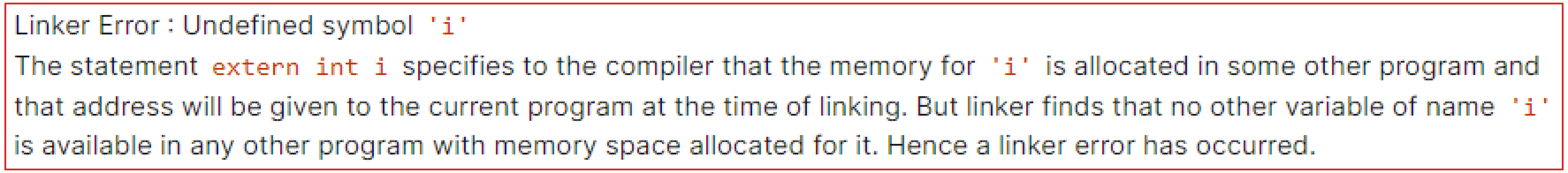


Ⓐ 2

Ⓑ 4

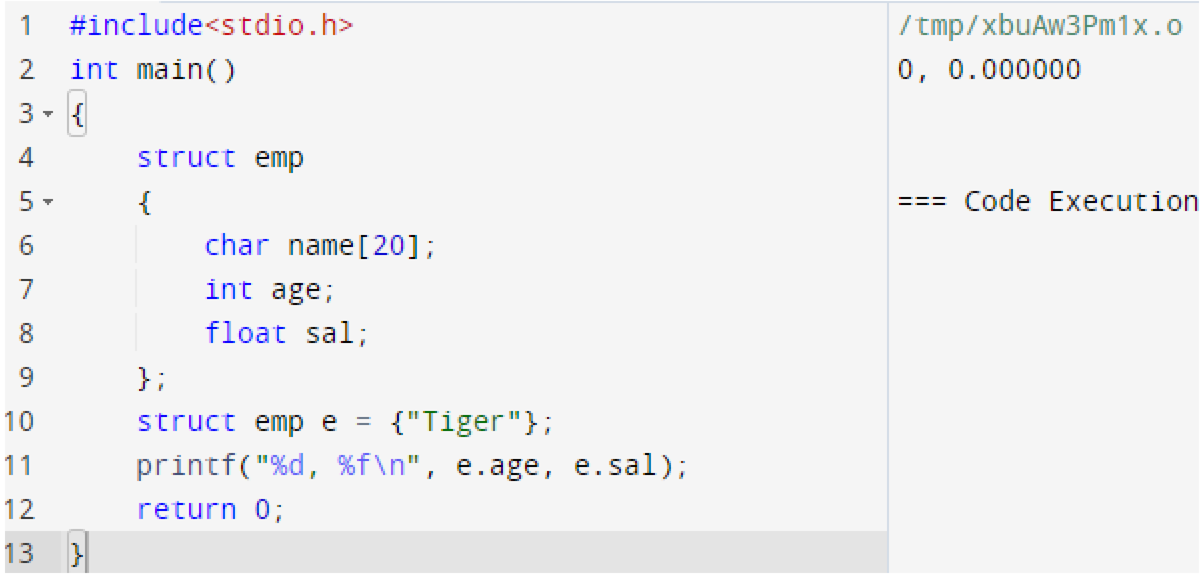
Ⓒ vary from compiler

Ⓓ Linker Error : Undefined symbol 'i' ✓



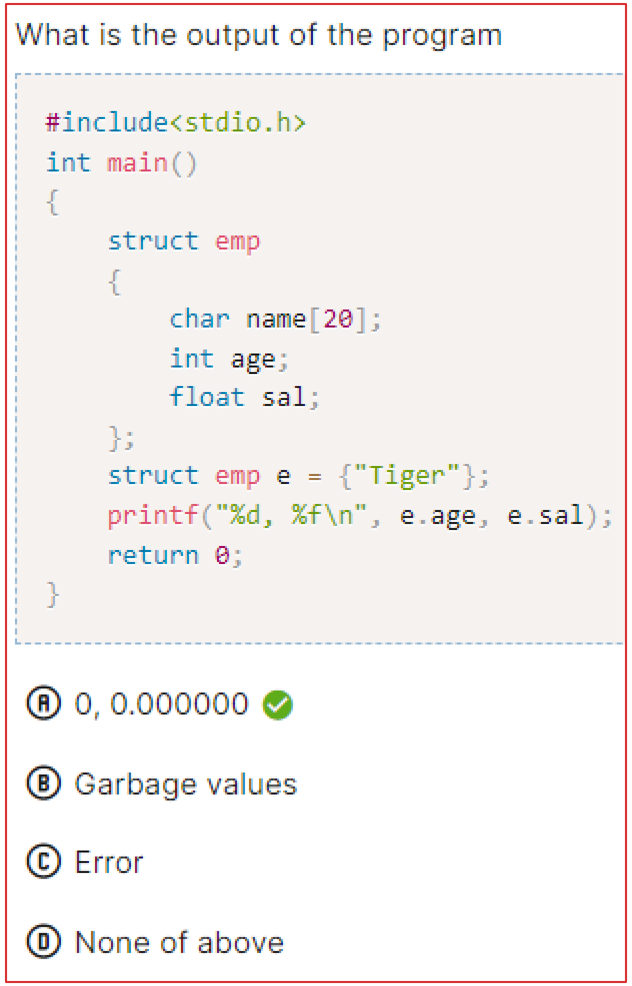
Linker Error : Undefined symbol 'i'

The statement `extern int i` specifies to the compiler that the memory for 'i' is allocated in some other program and that address will be given to the current program at the time of linking. But linker finds that no other variable of name 'i' is available in any other program with memory space allocated for it. Hence a linker error has occurred.



```c
#include<stdio.h>
int main()
{
    struct emp
    {
        char name[20];
        int age;
        float sal;
    };
    struct emp e = {"Tiger"};
    printf("%d, %f\n", e.age, e.sal);
    return 0;
}
```

```
/tmp/xbuAw3Pm1x.o
0, 0.000000


=== Code Execution
```



What is the output of the program

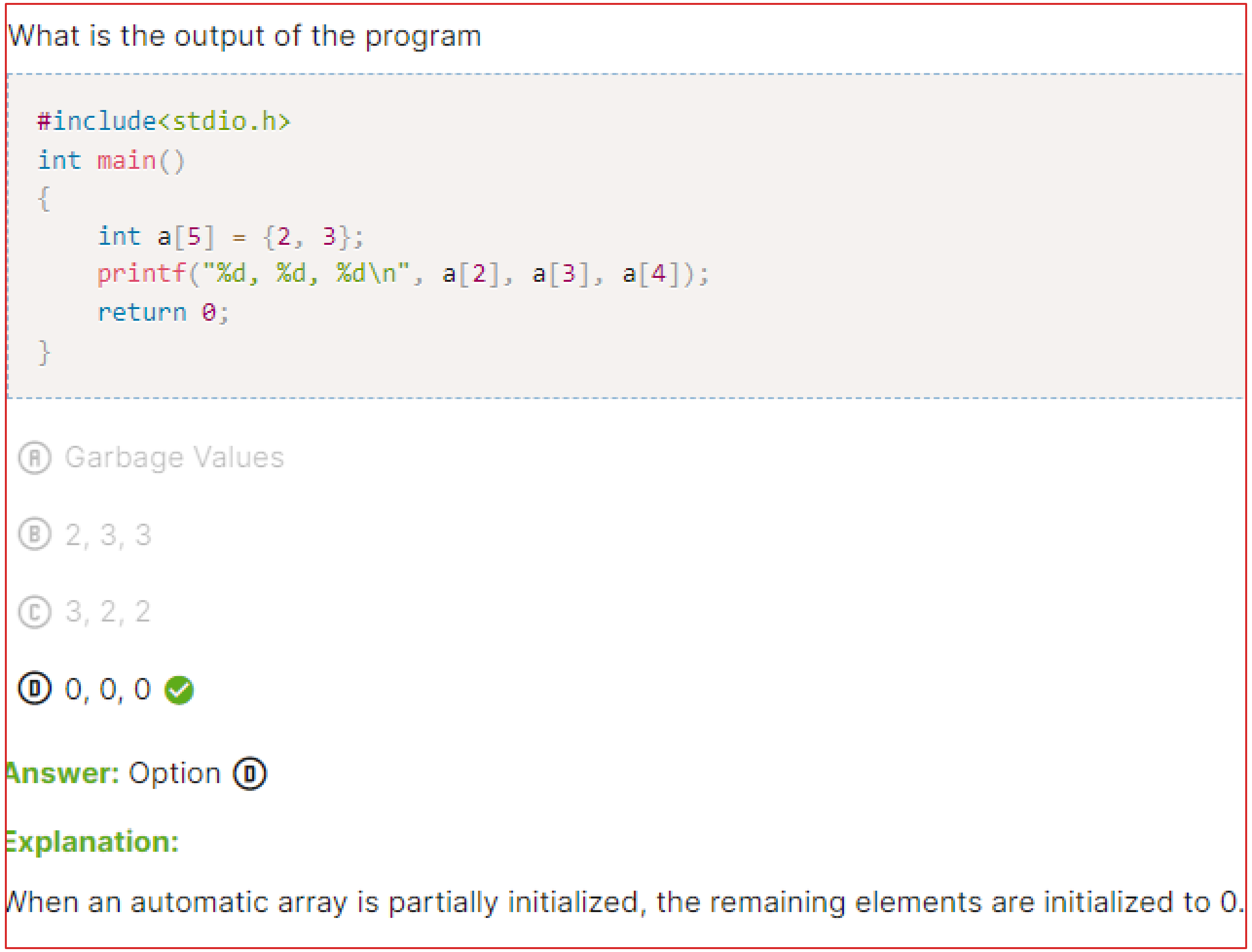
```c
#include<stdio.h>
int main()
{
    struct emp
    {
        char name[20];
        int age;
        float sal;
    };
    struct emp e = {"Tiger"};
    printf("%d, %f\n", e.age, e.sal);
    return 0;
}
```

Ⓐ 0, 0.000000 ✓

Ⓑ Garbage values

Ⓒ Error

Ⓓ None of above



What is the output of the program

```
#include<stdio.h>
int main()
{
    int a[5] = {2, 3};
    printf("%d, %d, %d\n", a[2], a[3], a[4]);
    return 0;
}
```

Ⓐ Garbage Values

Ⓑ 2, 3, 3

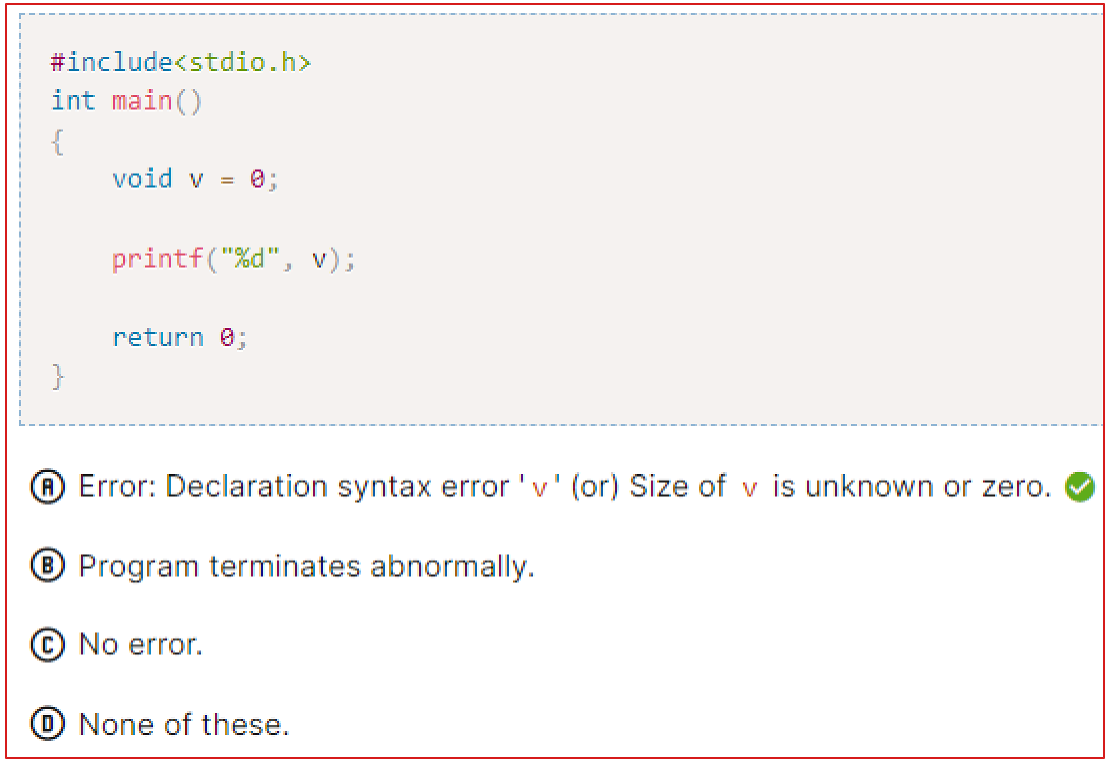
Ⓒ 3, 2, 2

Ⓓ 0, 0, 0 ✓

Answer: Option Ⓓ

Explanation:

When an automatic array is partially initialized, the remaining elements are initialized to 0.



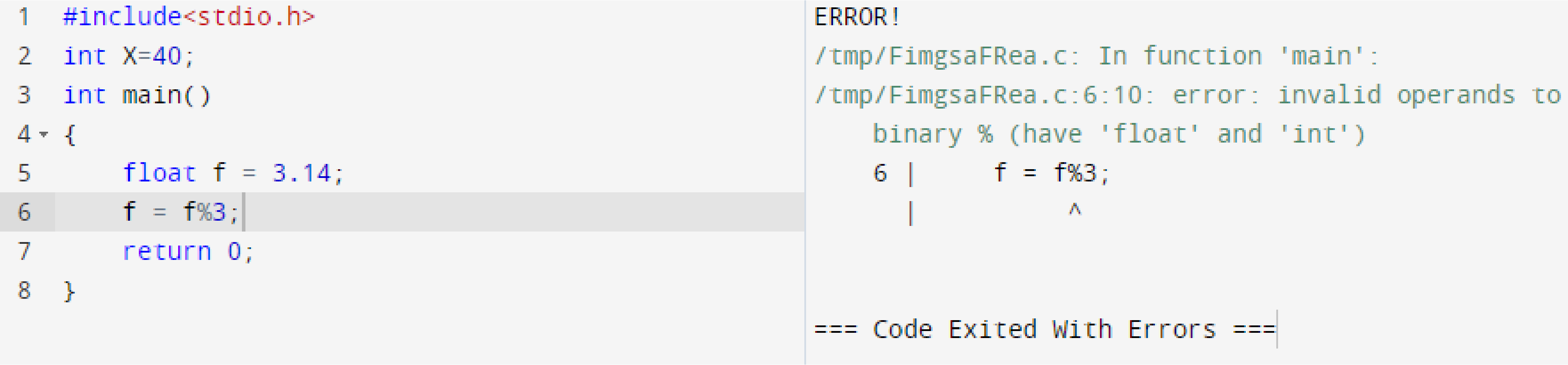
```
#include<stdio.h>
int main()
{
    void v = 0;

    printf("%d", v);

    return 0;
}
```

Ⓐ Error: Declaration syntax error 'v' (or) Size of v is unknown or zero. ✓

Ⓑ Program terminates abnormally.

Ⓒ No error.

Ⓓ None of these.

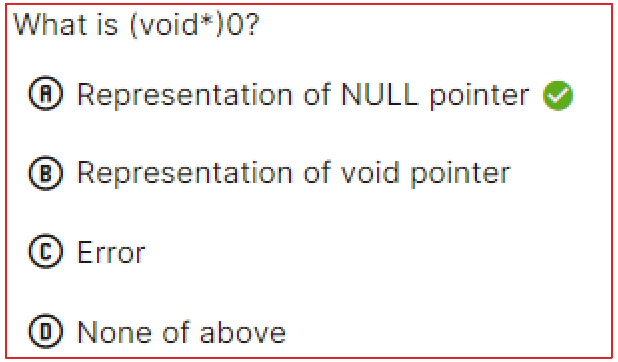


```c
#include<stdio.h>
int X=40;
int main()
{
    float f = 3.14;
    f = f%3;
    return 0;
}
```

```
ERROR!
/tmp/FimgsaFRea.c: In function 'main':
/tmp/FimgsaFRea.c:6:10: error: invalid operands to
    binary % (have 'float' and 'int')
    6 |      f = f%3;
      |         ^


=== Code Exited With Errors ===
```

`float a = 3.14; a = a%3;` gives "Illegal use of floating point" error.

The modulus (%) operator can only be used on integer types. We have to use `fmod()` function in math.h for float values.



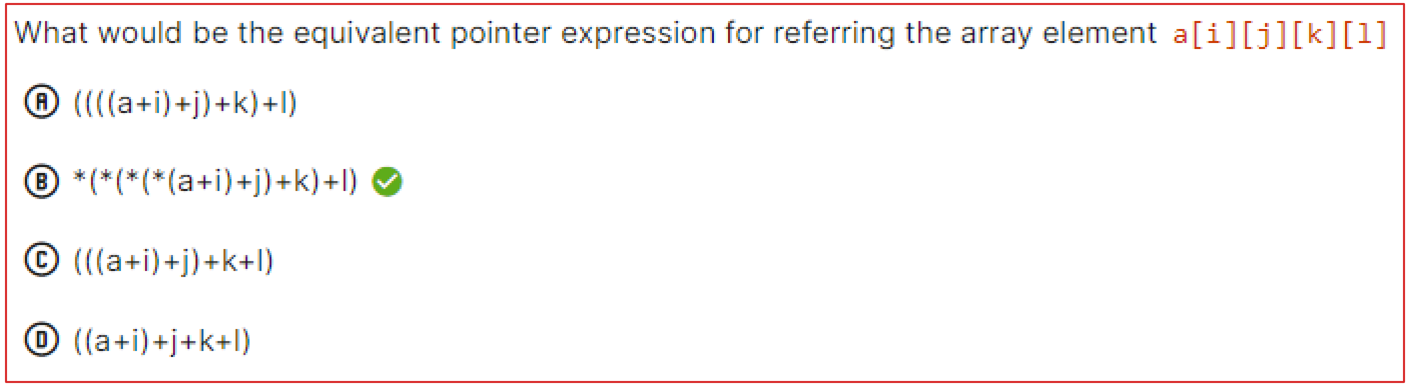
What is (void*)0?

Ⓐ Representation of NULL pointer ✓

Ⓑ Representation of void pointer

Ⓒ Error

Ⓓ None of above



What would be the equivalent pointer expression for referring the array element `a[i][j][k][l]`

Ⓐ ((((a+i)+j)+k)+l)

Ⓑ *(*(*(*(a+i)+j)+k)+l) ✓

Ⓒ (((a+i)+j)+k+l)

Ⓓ ((a+i)+j+k+l)

viettel

Theo cách của bạn

THE END

Thanks you for your listen !