WORLD UNIVERSITY RANKINGS
UNITED INTERNATIONAL UNIVERSITY
TOP RANKED
PRIVATE UNIVERSITY IN BANGLADESH
QS UNIVERSITY RANKINGS ASIA 2019
UNITED INTERNATIONAL UNIVERSITY
HAS BEEN RANKED ITS POSITION AMONG TOP 350 UNIVERSITIES IN ASIA.

Structured Programming Language CSI 121/CSE 1111 -Pointer-

Computer Science & Engineering (CSE)

United International University (UIU)

Introduction to Pointer

- Pointers are symbolic representation of addresses.

- Every variable is a memory location and every memory location has its address defined which can be accessed using ampersand (&) operator, which denotes an address in memory.

Syntax: datatype *var_name;

```
int *ptr; //ptr can point to an address
               which holds int data
```

Pointer

- A variable that contains the memory address of another variable
- Pointer type MUST match variable type
  - `int` pointer cannot point to a `float` variable

valid pointer declarations

- `int *ip; // pointer to an integer`
- `double *dp; // pointer to a double`
- `float *fp; // pointer to a float`
- `char *ch // pointer to a character`

Prints the address of the variables defined

```
#include <stdio.h>
int main () {
   int var1;
   char var2[10];
   printf("Address of var1 variable: %x
             \n", &var1 );
   printf("Address of var2 variable: %x
             \n", &var2 );
   return 0;
}
```

Output:

```
Address of var1 variable: bff5a400
Address of var2 variable: bff5a3f6
```

Pointer: Initialization or assignment

```
int a = 5;   // a contains value 5
int *p = &a; // p contains the address
of a

OR
int a = 5; // a contains value 5
int *p;    // pointer p is declared
p = &a;    // p contains the address
of a
```

Pointer: Initialization or assignment

```
int a = 5, b = 8;

int *p = &a;
int *q = &b;
```

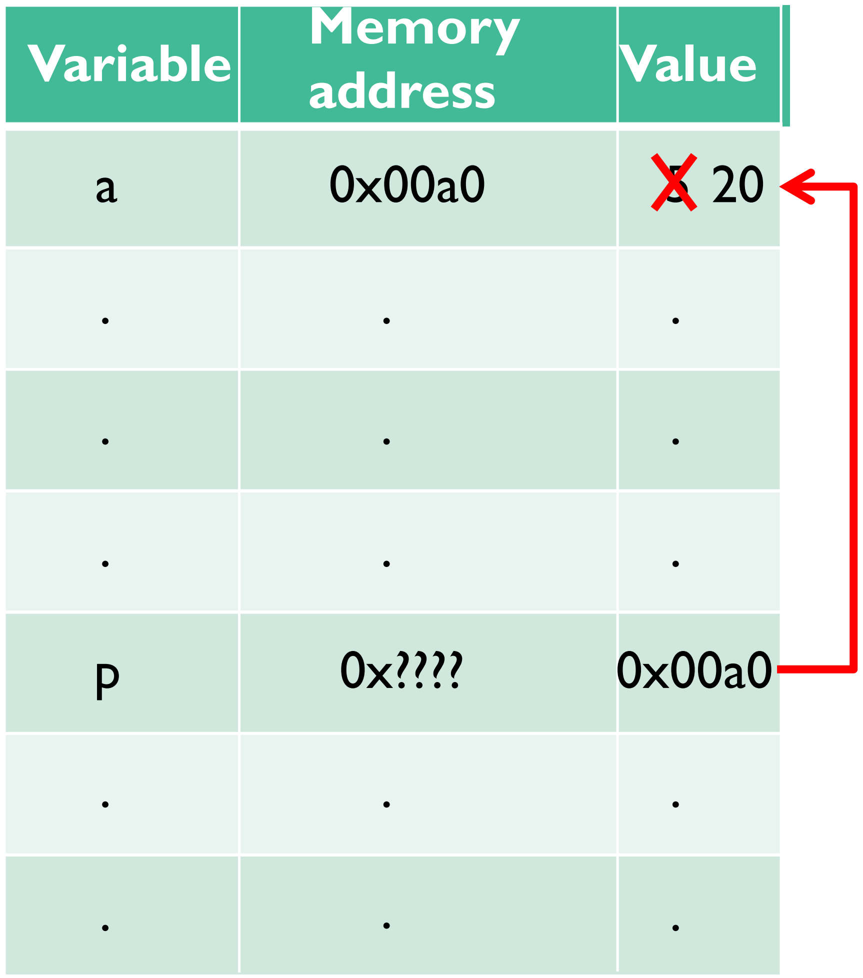
Variable	Memory address	Value
a	0x00a0	5
.	.	.
b	0x00b0	8
.	.	.
p	0x????	0x00a0
q	0x????	0x00b0
.		.

Pointer: Getting or manipulating value

```
int a = 5;
int *p = &a;

*p = 20;

// The following
line will
// print 20
printf("%d\n", a);
```



Variable	Memory address	Value
a	0x00a0	~~5~~ 20
.	.	.
.	.	.
.	.	.
p	0x????	0x00a0
.	.	.
.	.	.

How to Use Pointers?

There are a few important operations:

(a) Define a pointer variable,

(b) Assign the address of a variable to a pointer

(c) Access the value at the address available in the pointer variable.

This is done by using **unary operator * that returns the value of the variable** located at the address specified by its operand.

Example

```
#include <stdio.h>
int main () {
    int var = 20; /* variable declaration */
    int *ip; /* pointer variable */
    ip = &var; /* store address of var*/
    printf("Address of var variable: %x\n", &var );
          /* address stored in pointer variable */
    printf("Address stored in ip variable: %x\n", ip );
         /* access the value using the pointer */
    printf("Value of *ip variable: %d\n", *ip )
    return 0;
}
```

```
Address of var variable: bffd8b3c
Address stored in ip variable: bffd8b3c
Value of *ip variable: 20
```

Pointer of pointer/double pointer

```
int a = 5;      //a contains value 5
int *p = &a;    // p contains the
                     address of a
int **pp = &p; // pp contains the
                     address of p


printf("%d\n", **pp); // what will
                     be the output?
```

POINTER ARITHMETIC

Array, revisited

```
int array[10] = {5, 2, 3, 9, 10, 1,
7, 5, 4, 6};
int *p = &array[0];

printf("%d\n", array);
// what will be the output?
printf("%d\n", p);
// what will be the output?
```

Array as a pointer

```
int a[5] = {3, 2, 5,
1, 4};
```

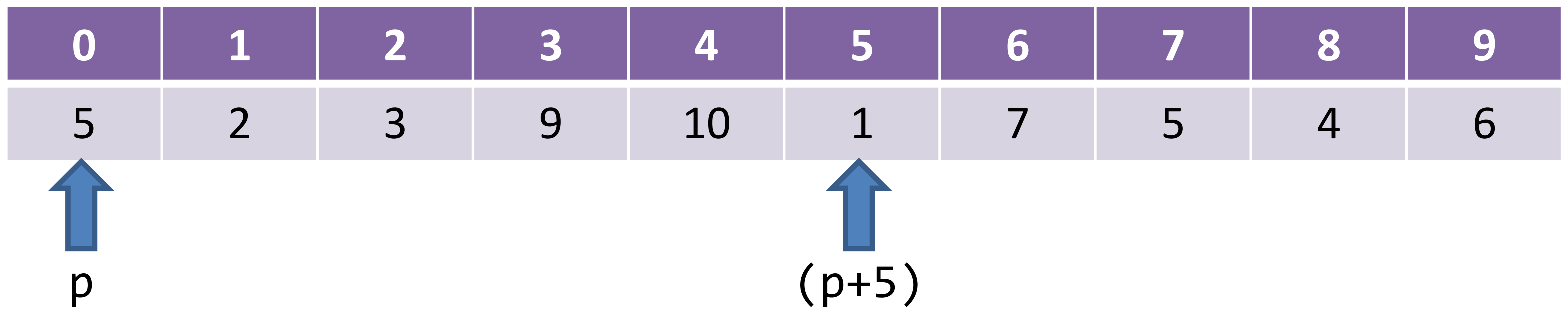
- The variable a contains the memory address of the first element of the array
- We can access the members of an array using pointer as well
- BEWARE: If you are not careful, you might end up in unauthorized memory addresses

Variable	Memory address	Value
a	0x00aa	0x0100
.	.	.
.	.	.
a[0]	0x0100	3
a[1]	0x0104	2
a[2]	0x0108	5
a[3]	0x010c	1
a[4]	0x0110	4
i	0x0114	234198
--	0x0118	21454
.	.	.
.	.	.

Pointer arithmetic

- To access array elements faster
  - Using pointers is faster than array indexing
- Four operators
  - + operator
  - - operator
  - ++ operator
  - -- operator

Pointer addition


0	1	2	3	4	5	6	7	8	9
5	2	3	9	10	1	7	5	4	6

```
int array[10] = {5, 2, 3, 9, 10, 1, 7,5,
4, 6};
int *p = array; // array points to 5

printf("%d", *(p+5));
// what does it print?
```

Pointer addition

```
int array[10] = {5, 2,
3, 9,10, 1, 7, 5,4,6};
int *p = array;
printf("%d",*(p+5));
```

Pointer	Index	Memory address	Value
p	0	0x0100 Base	5
p+1	1	0x0104 address	2
	2	0x0108	3
p+3	3	0x010c	9
	4	0x0110	10
p+5	5	0x0114 How to compute?	1
	6	0x0118	7
	7	0x011c	5
	8	0x0120	4
p+9	9	0x0124	6
	10	0x0128	?
	11	0x012c	?

Pointer addition

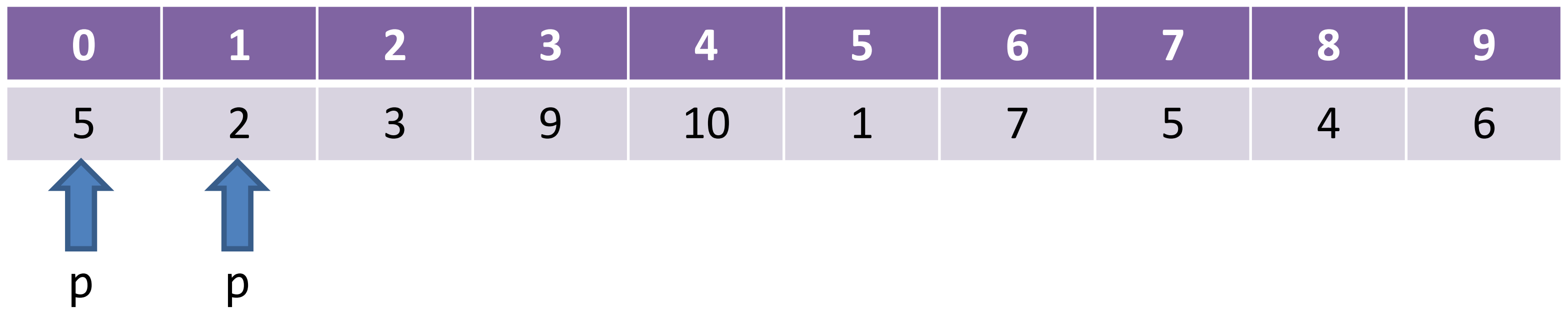
- If we add 1 to a pointer, we are actually adding the space of one variable
- If we have 4-byte integer variable, adding 1 adds 4 to the memory address
- New address = Base address + Variable size * i
- Example: Let the base address of p be 0x0100
  - p+1: 0x0100+4*1 = 0x0104
  - p+3: 0x0100+4*3 = ~~0x0112~~ 0x010c
    - Given address is in hexadecimal!!!
    - Use a calculator to be safe
  - p+5: Try yourself!!!

Problem

Write a C program that prints the content of a given array.

Use pointer addition instead of array indexing.

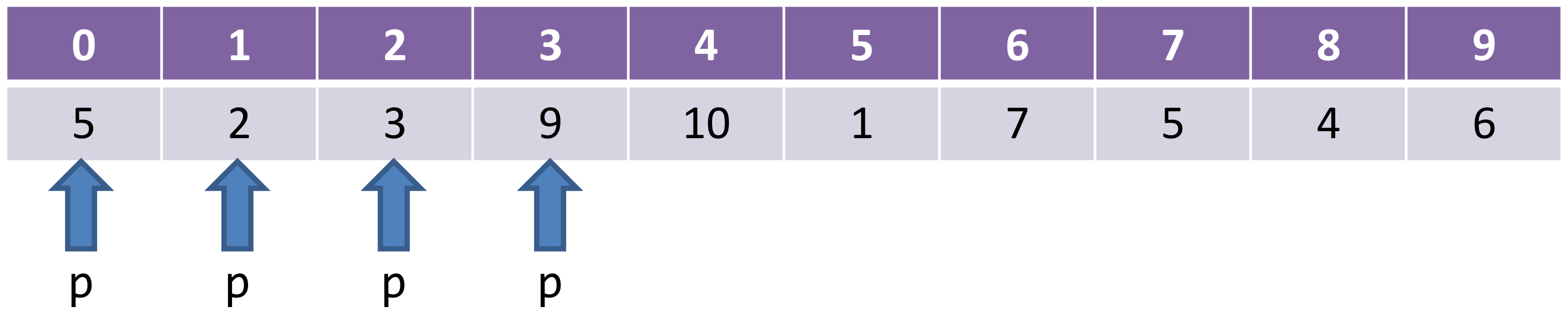
Pointer increment



0	1	2	3	4	5	6	7	8	9
5	2	3	9	10	1	7	5	4	6

```
int array[10] = {5, 2, 3, 9, 10, 1, 7, 5,
4, 6};
int *p = array;   // array points to 5
p++;              // where does p point
to?
printf("%d", *p); // what does it print?
```

Pointer increment



0	1	2	3	4	5	6	7	8	9
5	2	3	9	10	1	7	5	4	6

```
int array[10] = {5, 2, 3, 9, 10, 1, 7, 5, 4,
6};
int *p = array;    // array points to 5
p++;               // where does p point to?
p++;               // where now?
p++;               // where now?
```

Pointer increment

```
int array[10] = {5, 2, 3,
9,10, 1, 7, 5, 4, 6};
int *p = array;
p++;
p++;
p++;
```

p initially

p after line 3

p after line 4

p after line 5

CAUTION: Do NOT access from here

Index	Memory address	Value
0	0x0100	5
1	0x0104	2
2	0x0108	3
3	0x010c	9
4	0x0110	10
5	0x0114	1
6	0x0118	7
7	0x011c	5
8	0x0120	4
9	0x0124	6
10	0x0128	?
11	0x012c	?

Cons of using C pointer

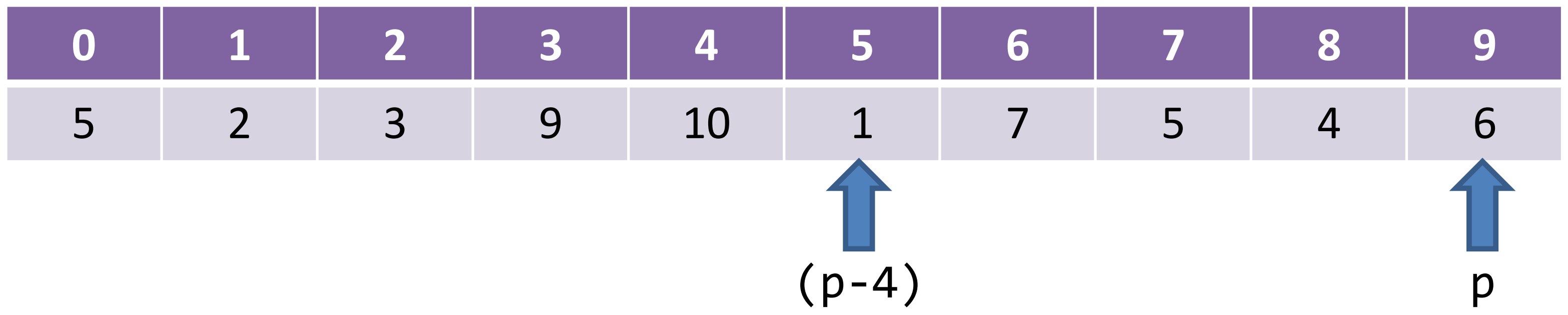
- C pointers are not secure
- They can reach virtually ANY memory address
  - Both in the programs own memory space and outside
  - May cause replacement of important data
  - May cause program to crash
- We manually make sure that the pointer does not try to access unauthorized memory spaces

Problem

Write a C program that prints the content of a given array.

Use pointer increment instead of array indexing.

Pointer subtraction



0	1	2	3	4	5	6	7	8	9
5	2	3	9	10	1	7	5	4	6

```
int array[10] = {5, 2, 3, 9, 10, 1, 7, 5, 4,
6};
int *p = &array[9]; // array points to 6

printf("%d", *(p-4)); // what does it print?
```

Pointer subtraction

```
int array[10] = {5, 2,
9, 10, 1, 7, 5, 4, 6};
int *p = &array[9];
printf("%d", *(p-4));
```

Pointer	Index	Memory address	Value
p-9	0	0x0100	5
	1	0x0104	2
	2	0x0108	3
p-6	3	0x010c	9
	4	0x0110	10
p-4	5	0x0114	1
	6	0x0118	7
	7	0x011c	5
p-1	8	0x0120	4
p	9	0x0124	6
	10	0x0128	?
	11	0x012c	?

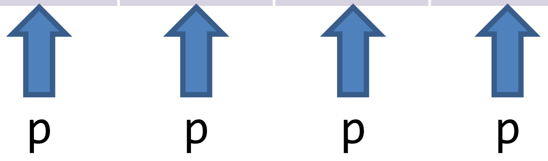
Problem

Write a C program that prints the content of a given array in reverse order.

Use pointer subtraction instead of array indexing.

Pointer decrement

0	1	2	3	4	5	6	7	8	9
5	2	3	9	10	1	7	5	4	6



```
int array[10] = {5, 2, 3, 9, 10, 1, 7, 5, 4,
6};
int *p = &array[9];   // array points to 5
p--;                 // where does p point to?
p--;                 // where now?
p--;                 // where now?
```

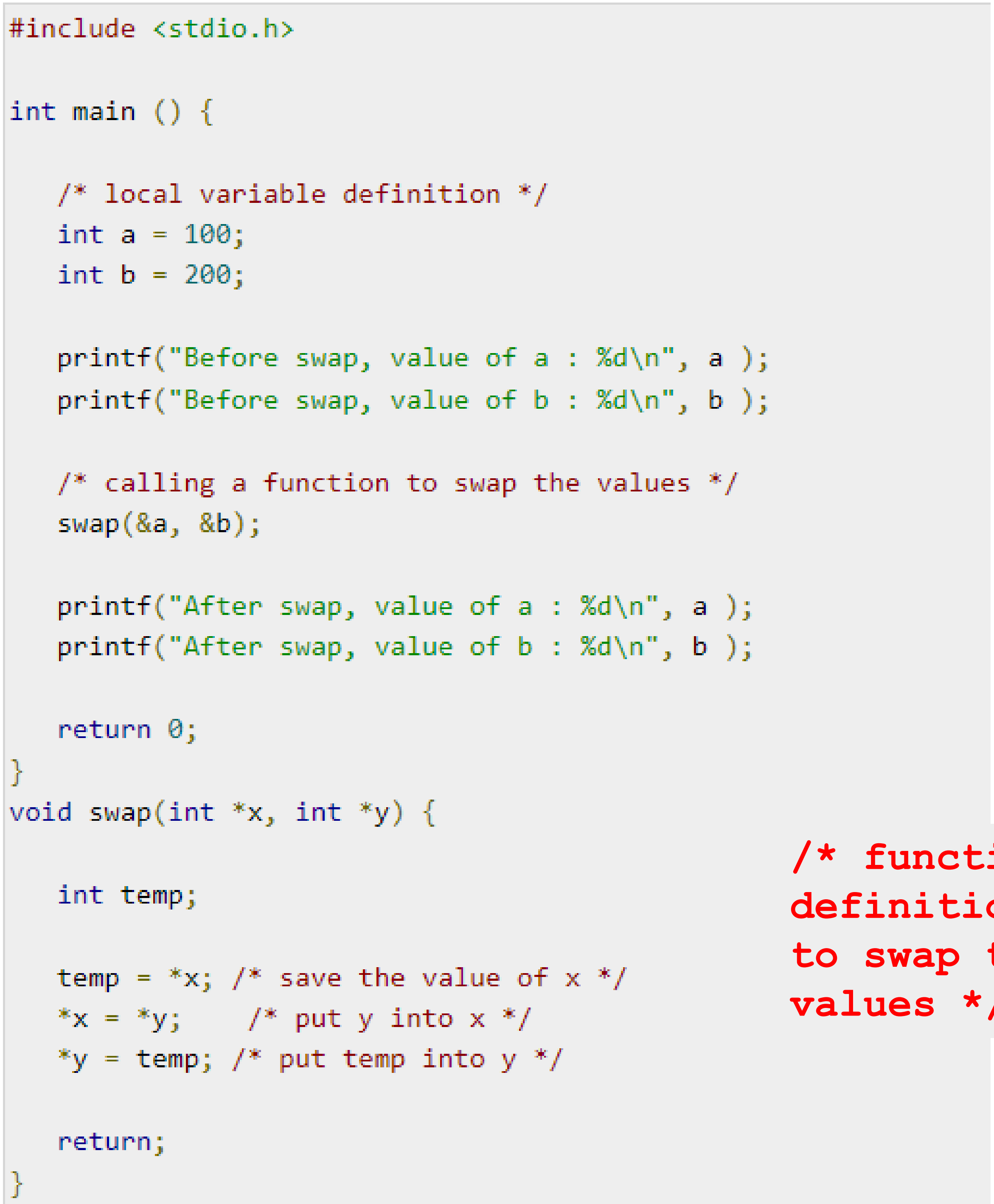
Pointer increment

```
int array[10] = {5, 2,
3, 9,10, 1, 7, 5, 4, 6};
int *p = &array[9];
p--;
p--;
p--;
```

	Index	Memory address	Value
	0	0x0100	5
	1	0x0104	2
	2	0x0108	3
	3	0x010c	9
	4	0x0110	10
p after line 5 ⇨	5	0x0114	1
p after line 4 ⇨	6	0x0118	7
p after line 3 ⇨	7	0x011c	5
p initially ⇨	8	0x0120	4
	9	0x0124	6
	10	0x0128	?
	11	0x012c	?

Passing Pointer to a Function

- When we pass a pointer as an argument instead of a variable then the address of the variable is passed instead of the value.
- So any change made by the function using the pointer is permanently made at the address of passed variable.
- This technique is known as **call by reference**.



```c
#include <stdio.h>

int main () {

   /* local variable definition */
   int a = 100;
   int b = 200;

   printf("Before swap, value of a : %d\n", a );
   printf("Before swap, value of b : %d\n", b );

   /* calling a function to swap the values */
   swap(&a, &b);

   printf("After swap, value of a : %d\n", a );
   printf("After swap, value of b : %d\n", b );

   return 0;
}
void swap(int *x, int *y) {

   int temp;

   temp = *x; /* save the value of x */
   *x = *y;    /* put y into x */
   *y = temp; /* put temp into y */

   return;
}
```

/* function definition to swap the values */

?

Thank You

Example

```
#include <stdio.h>
void salaryhike(int *var, int b){
      *var = *var + b;
}
int main() {
      int salary=0, bonus=0;
      printf("Enter the employee current salary:");
      scanf("%d", &salary);
      printf("Enter bonus:");
      scanf("%d", &bonus);
      salaryhike(&salary, bonus);
      printf("Final salary: %d", salary);
      return 0;
}
```