Pointers in C

Operator

Decision making

Loops

Arrays – 2D, 3D

Basics of Functions

Pointers

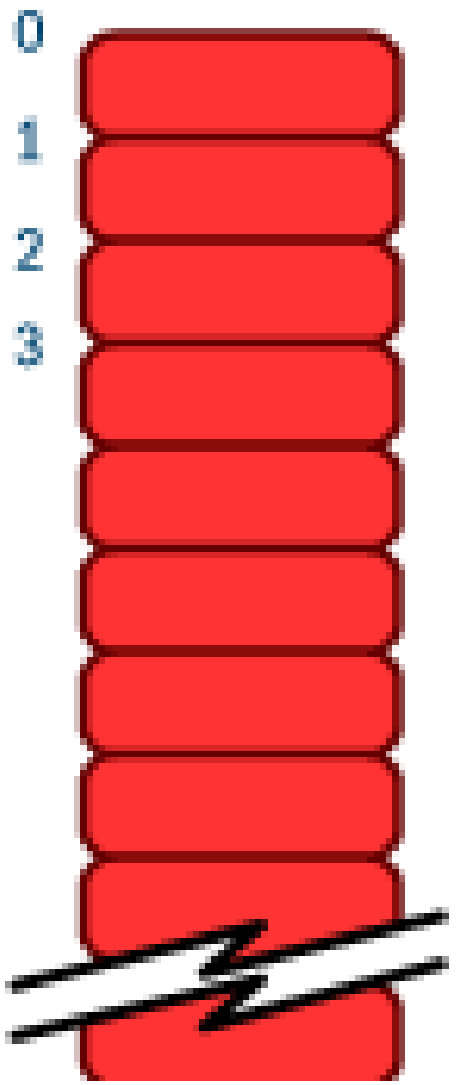
Arrays and Pointer, Strings

Functions, parameter passing, array passing, return multiple things

Structures –

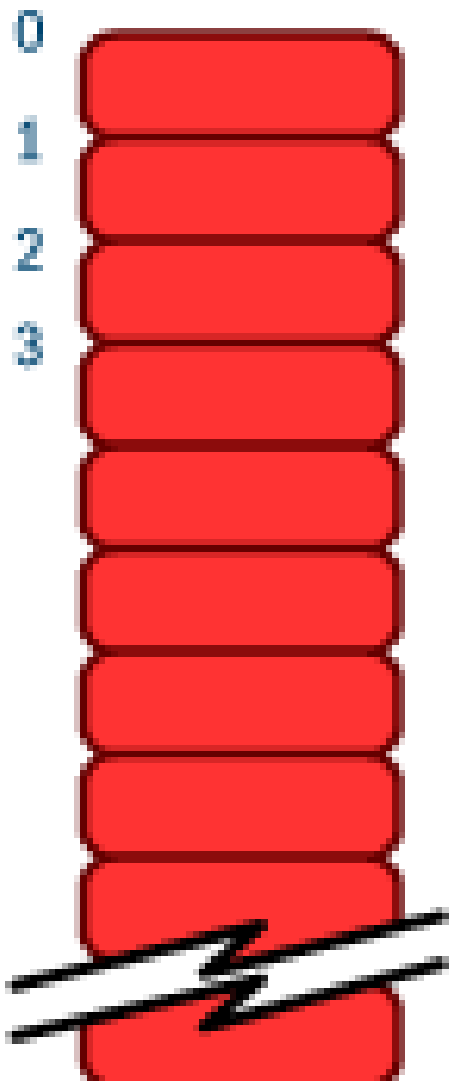
Searching and Sorting – on numbers and strings

Linked List



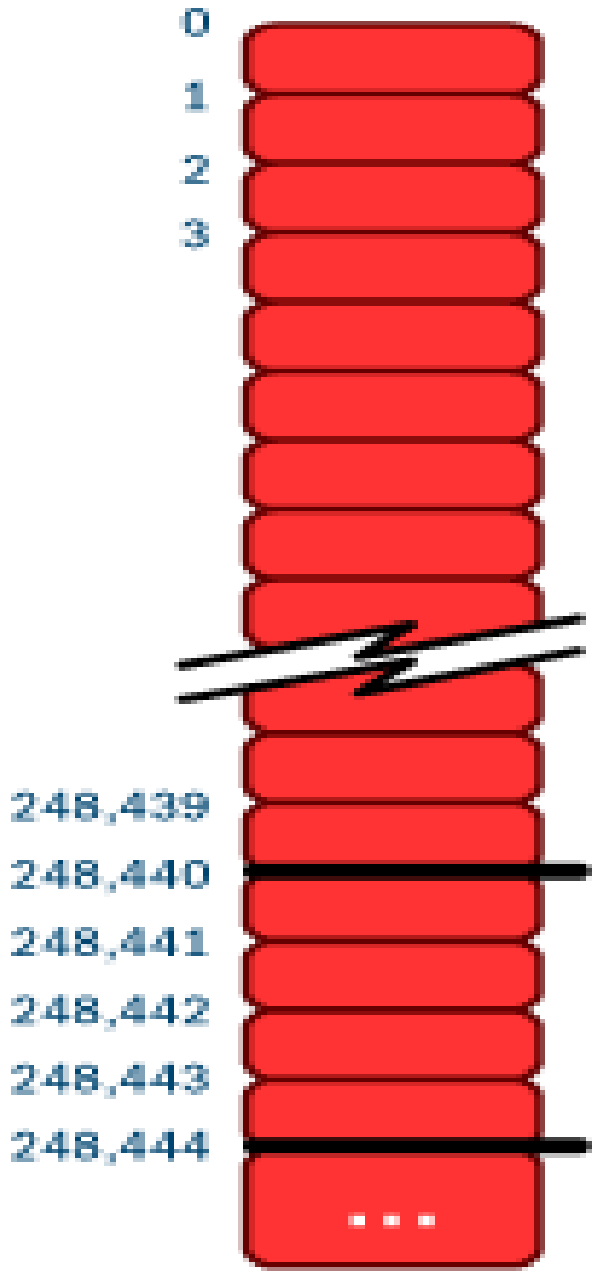
Memory is an

array of bytes



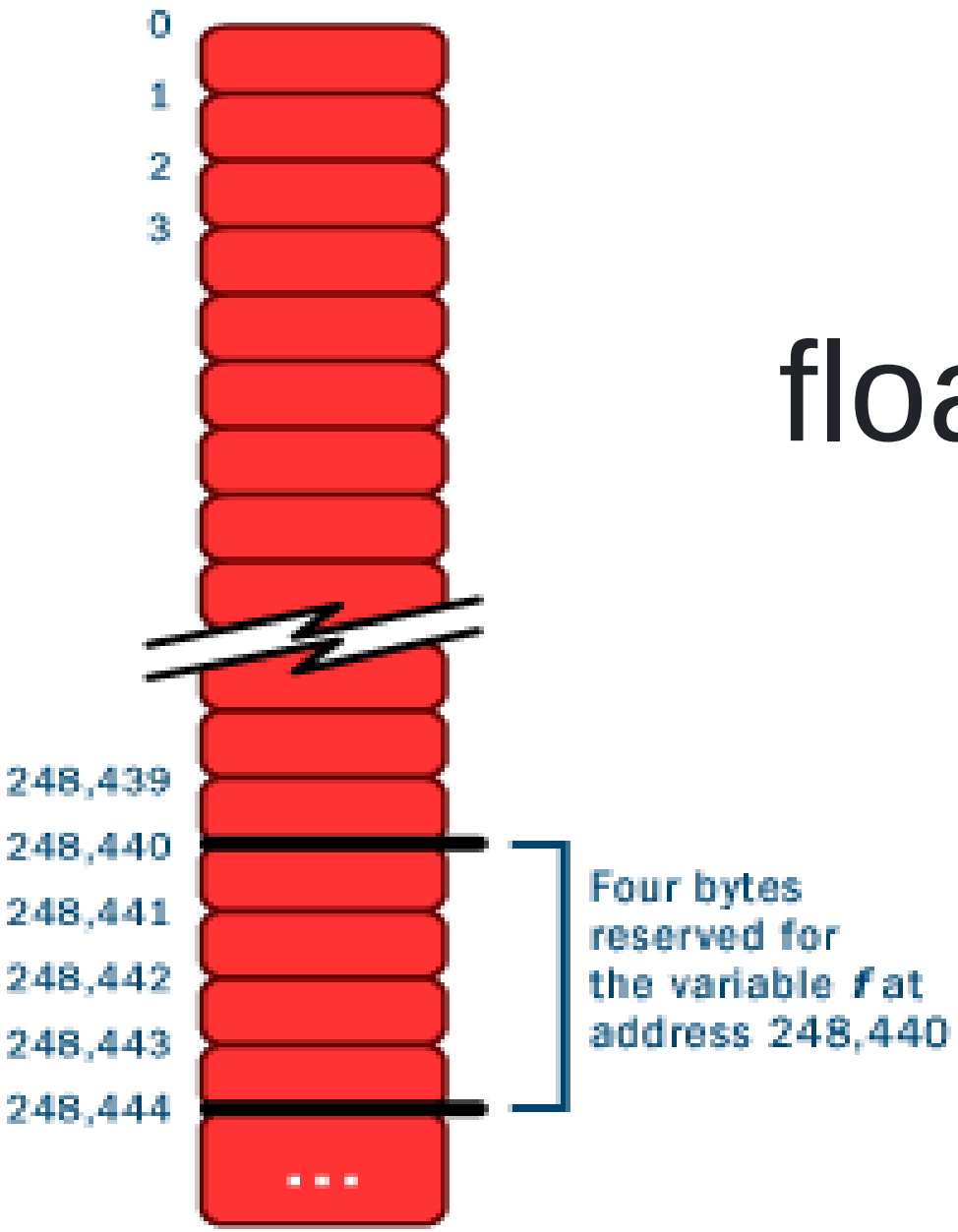
Memory is an

array of bytes



Memory is an

array of bytes


0
1
2
3
248,439
248,440
248,441
248,442
248,443
248,444
Four bytes reserved for the variable f at address 248,440

float f;

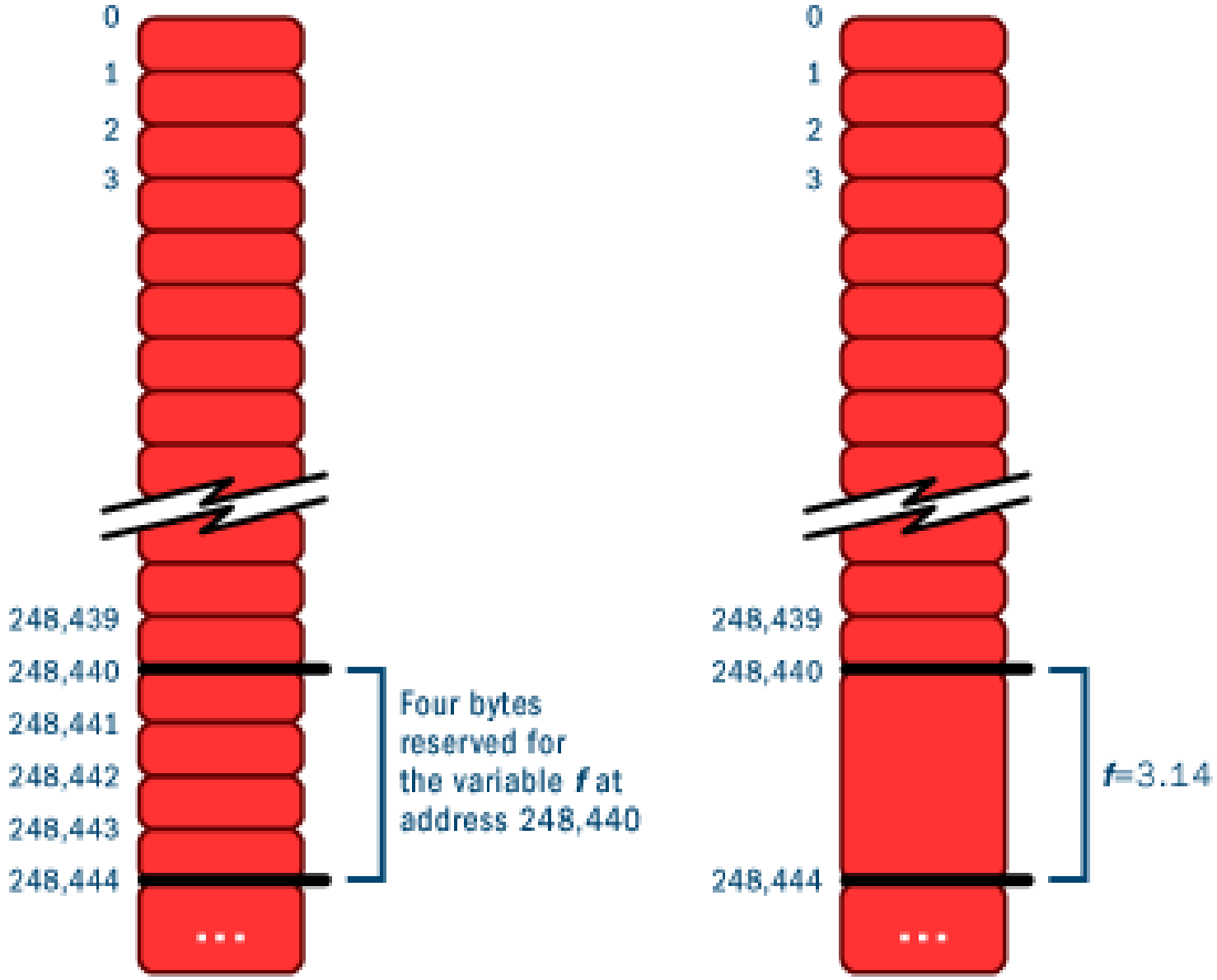
- How much **memory** you have in your mobile ?
- 12 GB – How many bytes you have in the RAM ??

- 2^{10} = 1024 Approx 1000 = 10^3
- ….
- 12 X 10^9 = Roughly = 12 000 000 000
- 12 X 2^{30} = 12 884 901 888
- Last byte number = 12 X 2^{30} - 1

While you think of the variable **f**,

the computer thinks of a specific address in memory (for example, 248,440).

Consider - f = 3.14;
Inside computer "**Load the value 3.14 into memory location 248,440**"


0
1
2
3
248,439
248,440
248,441
248,442
248,443
248,444
Four bytes reserved for the variable f at address 248,440
0
1
2
3
248,439
248,440
248,444
f=3.14

One possible side effect

```
int i, s[4], t[4], u=0;

for (i=0; i<=4; i++)
{
        s[i] = i;
        t[i] =i;
}
printf("s:t\n");
for (i=0; i<=4; i++)
        printf("%d:%d\n", s[i], t[i]);
printf("u = %d\n", u);
```

The output that you see from the program will **probably** look like this:

s:t

0:4

1:1

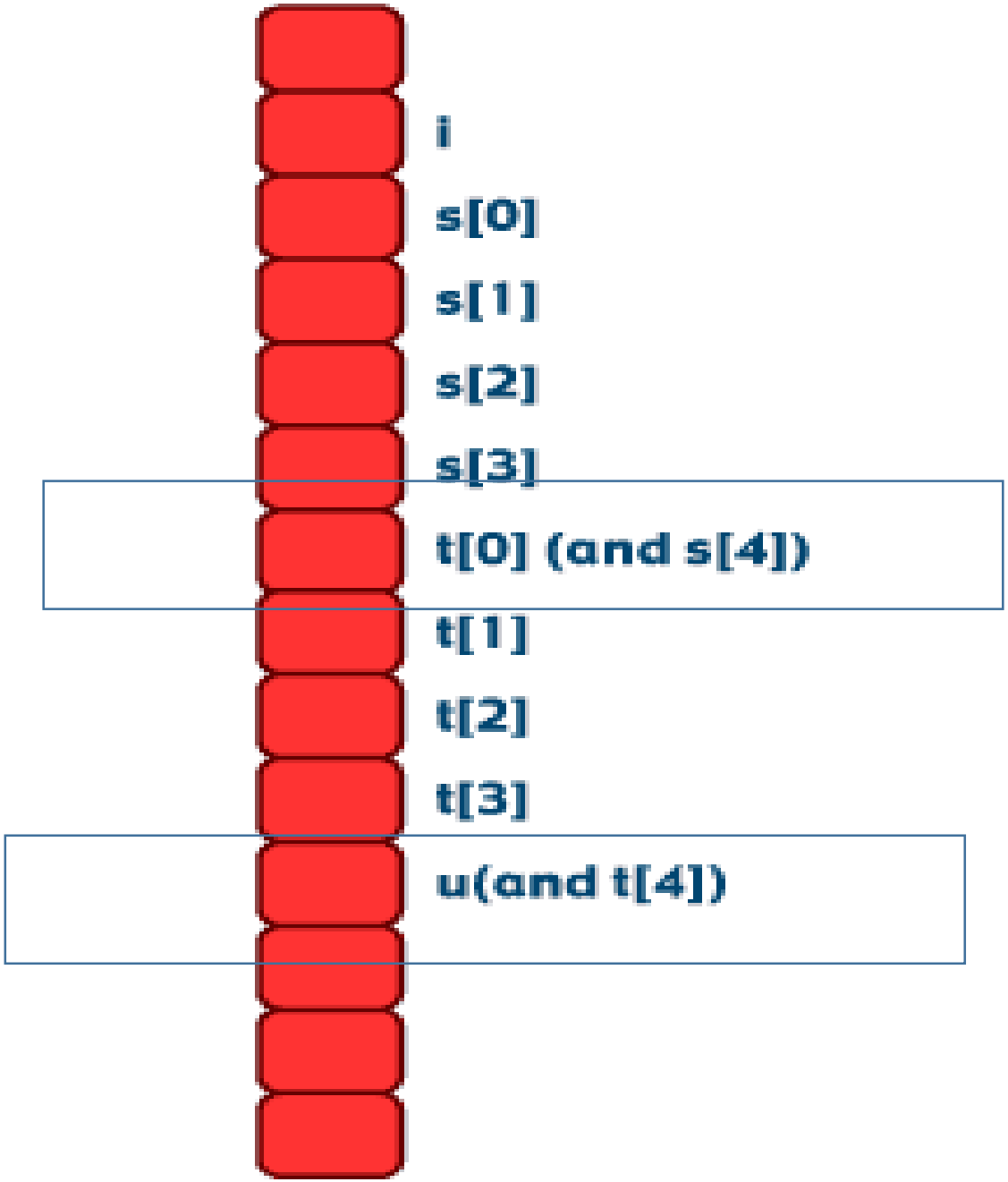
2:2

3:3

4:4

u = 4


i
s[0]
s[1]
s[2]
s[3]
t[0] (and s[4])
t[1]
t[2]
t[3]
u(and t[4])


i
s[0]
s[1]
s[2]
s[3]
t[0] (and s[4])
t[1]
t[2]
t[3]
u(and t[4])

How to obtain the address of a variable ?

int i

How to obtain the address of a variable ?

int i

&i – is the address of the variable i

How to print the address ?

```
#include <stdio.h>
int main()
{
  int var = 5;
  printf("var: %d\n", var);

  // Notice the use of & before var
  printf("address of var: %p", &var);   // %u
  return 0;
}
```

How to print the address ?

var: 5

address of var: 2686778

Note: You “may” get a different address when you run the above code.

Pointers

Addresses are fundamentally integers

They are treated specially – since size of a variable depends on the type of the variable

Special variables that are used to store the addresses of a variable is called pointer variables

Pointers (Pointer variables)

```
int* p;
```

Here, we have declared a pointer p of int type.

Pointers

```
int *p;
```

Here, we have declared a pointer p of int type.

```
int *p1;
int *p2;
```

Example

```
int *pc, c;
c = 5;
pc = &c;
```

5 is assigned to c,
Address of c is assigned to pc

Example

```
#include <stdio.h>

int main()
{
    int i,j;
    int *p;   /* a pointer to an integer */
    printf("%d %d\n", p, &i);
    p = &i;
    printf("%d %d\n", p, &i);
    return 0;
}
```

Explanation

The code tells the compiler to print out the address held in p, along with the address of i.

The variable p starts off with some crazy value or with 0.

The address of i is generally a large value.

Sample output

0 2147478276

2147478276 2147478276

(The address of i is 2147478276. Once the statement p = &i; has been executed, p contains the address of i.)

Get Value Pointed by Pointers

* Operator

Check the following code -

```
int *pc, c;
c = 5;
pc = &c;
printf("%d", *pc);
```

Get Value Pointed by Pointers

* Operator

Check the following code -

```
int* pc, c;
c = 5;
pc = &c;
printf("%d", *pc);   // Output: 5
```

Explanation

Here, the address of c is assigned to the pc pointer.

To get the value stored in that address, we used *pc.

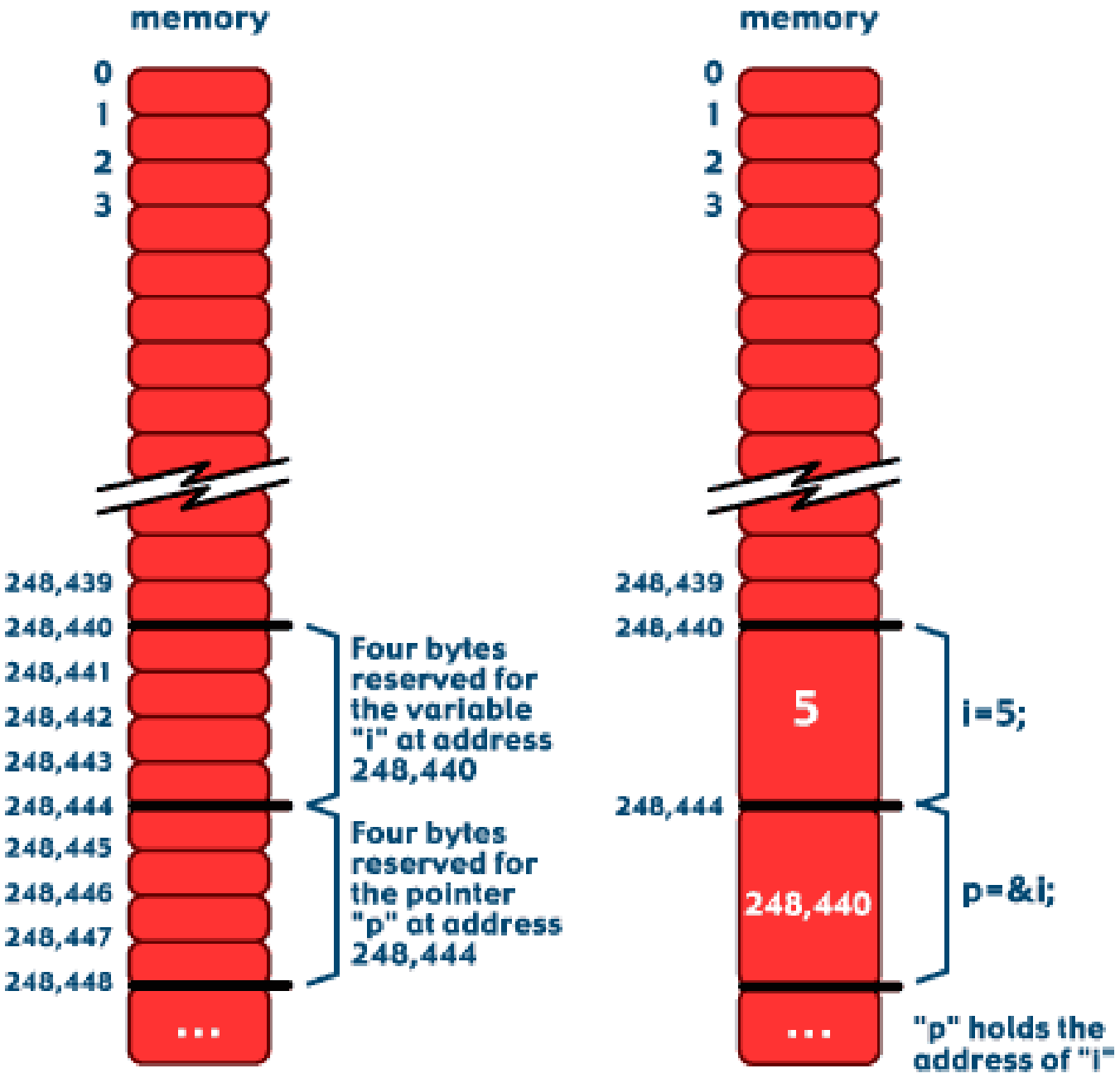
pc is a pointer, not *pc. *pc = &c → does not work

* is called the **dereference operator** → operates on a pointer and gives the value stored in that pointer

Example

```
#include <stdio.h>

int main()
{
   int i;
   int *p;   /* a pointer to an integer */
   p = &i;
   *p=5;
   printf("%d %d\n", i, *p);
   return 0;
}
```


memory
0
1
2
3
248,439
248,440
248,441
248,442
248,443
248,444
248,445
248,446
248,447
248,448
...
Four bytes reserved for the variable "i" at address 248,440
Four bytes reserved for the pointer "p" at address 248,444
memory
0
1
2
3
248,439
248,440
5
248,444
248,440
...
i=5;
p=&i;
"p" holds the address of "i"

How to print the address ?

```
#include <stdio.h>
int main()
{
  int var = 5;
  int *p;
  int **q;
  printf("var: %d\n", var);
  p = &var;
  q = &p;
  // Notice the use of & before var
  printf("address of var: %p", &var);   // %u
  printf("address of p: %p", &p);   // %u
  return 0;
}
```

Example

```
int *pc, c;
c = 5;
pc = &c;
c = 1;
printf("%d", c);
printf("%d", *pc);
```

Example

```
int *pc, c;
c = 5;
pc = &c;
c = 1;
printf("%d", c);    // Output: 1
printf("%d", *pc);  // Ouptut: 1
```

Example

```
int* pc, c;
c = 5;
pc = &c;
*pc = 1;
printf("%d", *pc);  // Ouptut: 1
printf("%d", c);    // Output: 1
```

Example

```
int *pc, c, d;
c = 5;
d = -15;

pc = &c; printf("%d", *pc); // Output: 5
pc = &d; printf("%d", *pc); // Ouptut: -15
```

Example

```
#include <stdio.h>
int main()
{
   int* pc, c;

   c = 22;
   printf("Address of c: %p\n", &c);
   printf("Value of c: %d\n\n", c);  // 22
```

Example

```
pc = &c;
printf("Address stored in pointer pc: %p\n", pc);
printf("Content of pointer pc: %d\n\n", *pc); // 22

c = 11;
printf("Address stored in pointer pc: %p\n", pc);
printf("Content of pointer pc: %d\n\n", *pc); // 11
```

Example

```
*pc = 2;
   printf("Address of c: %p\n", &c);
   printf("Value of c: %d\n\n", c); // 2
   return 0;
}
```

Sample output

Address of c: 2686784
Value of c: 22

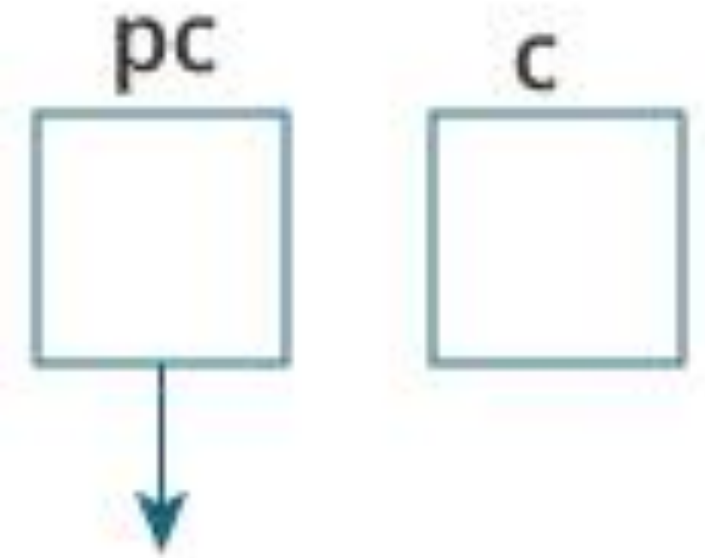
Address stored in pointer pc: 2686784
Content of pointer pc: 22

Address stored in pointer pc: 2686784
Content of pointer pc: 11

Address of c: 2686784
Value of c: 2

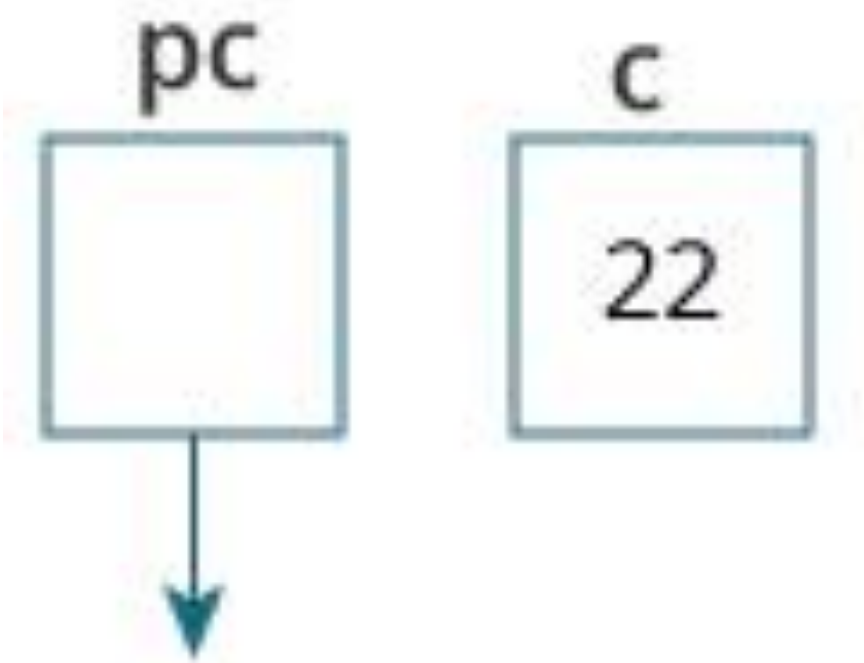
Explanation with diagram

int *pc, c;



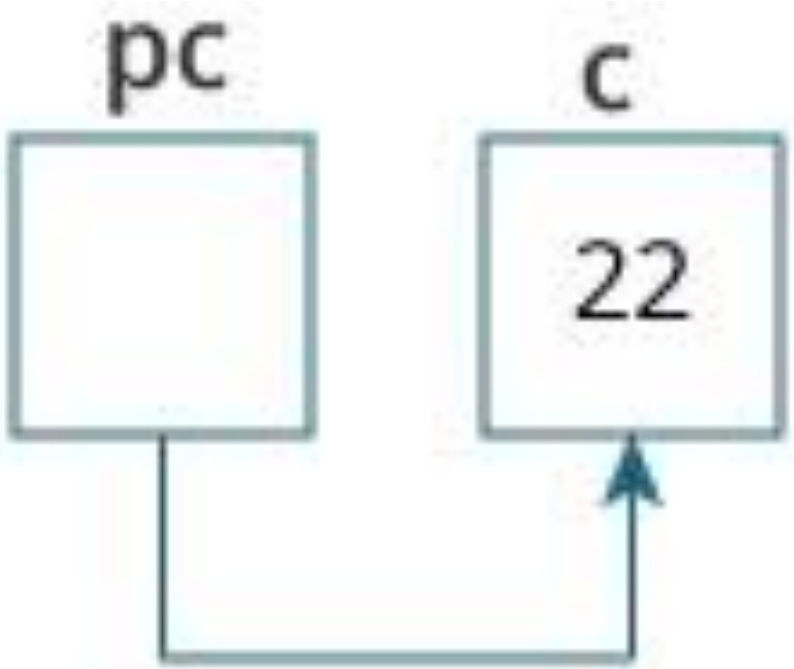
Explanation with diagram

c = 22;



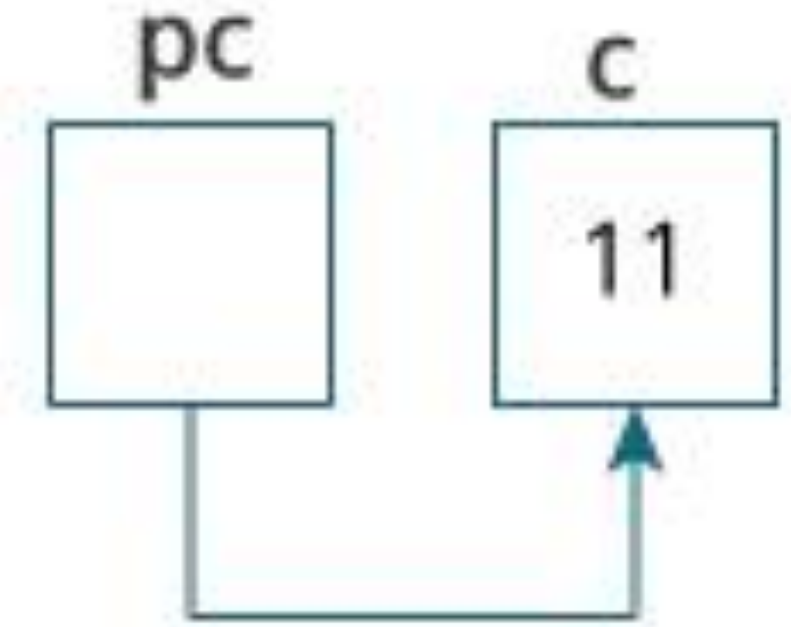
Explanation with diagram

pc = &c;



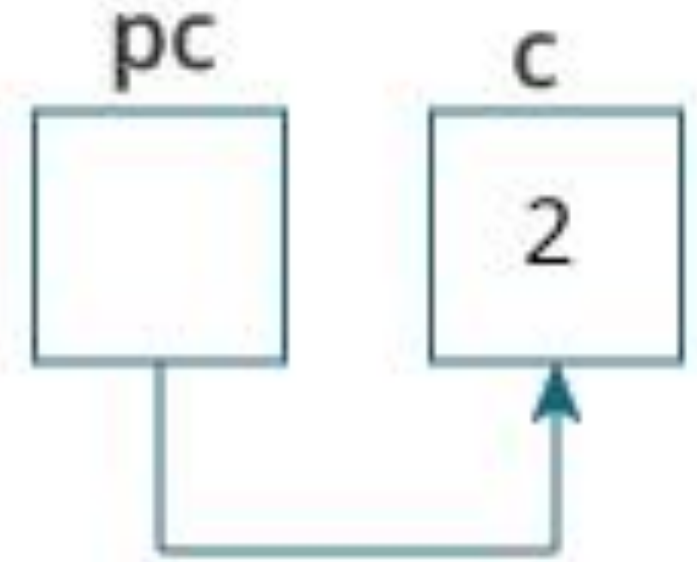
Explanation with diagram

c = 11;



Explanation with diagram

*pc = 2;



Common mistakes

```
int c, *pc;

// pc is address but c is not
pc = c; // Error

// &c is address but *pc is not
*pc = &c; // Error
```

Common mistakes

```
// both &c and pc are addresses
pc = &c;

// both c and *pc values
*pc = c;
```

Confusion

```
#include <stdio.h>
int main() {
   int c = 5;
   int *p = &c;

  printf("%d", *p);  // 5
   return 0;
}
```

Why didn't we get an error when using int *p = &c;?

int *p = &c;

is equivalent to

int *p:

p = &c;

In both cases, we are creating a pointer p (not *p) and assigning &c to it.

To avoid this confusion, we can use the statement like this:

int* p = &c;

Relationship Between Arrays and Pointers

```
#include <stdio.h>
int main() {
   int x[4];
   int i;
   for(i = 0; i < 4; ++i) {
      printf("&x[%d] = %p\n", i, &x[i]);
   }
   printf("Address of array x: %p", x);
   return 0;
}
```

Relationship Between Arrays and Pointers

&x[0] = **1450734448**

&x[1] = 1450734452

&x[2] = 1450734456

&x[3] = 1450734460

Address of array x: **1450734448**

X and &X[0] are same

So, X is a pointer to X[0]

If x is the name of an array then its constant pointer variable.

You cannot change the content of x;

x = &i; --- This does not work – Error.

Key things to be notes

There is a difference of 4 bytes between two consecutive elements of array x.

It is because the size of int is 4 bytes (in gcc)

The address of &x[0] and x is the same → It's because the variable name x points to the first element of the array.

Addition of an integer with a pointer

Imagine x is a pointer of integer type

```
int *x;
int a = 5;
x = &a;
x = x+1;
print( *x) – what it will print ??
```

Are we adding one byte OR something else ??

Addition of an integer with a pointer

Imagine x is a pointer of integer type

int *x;

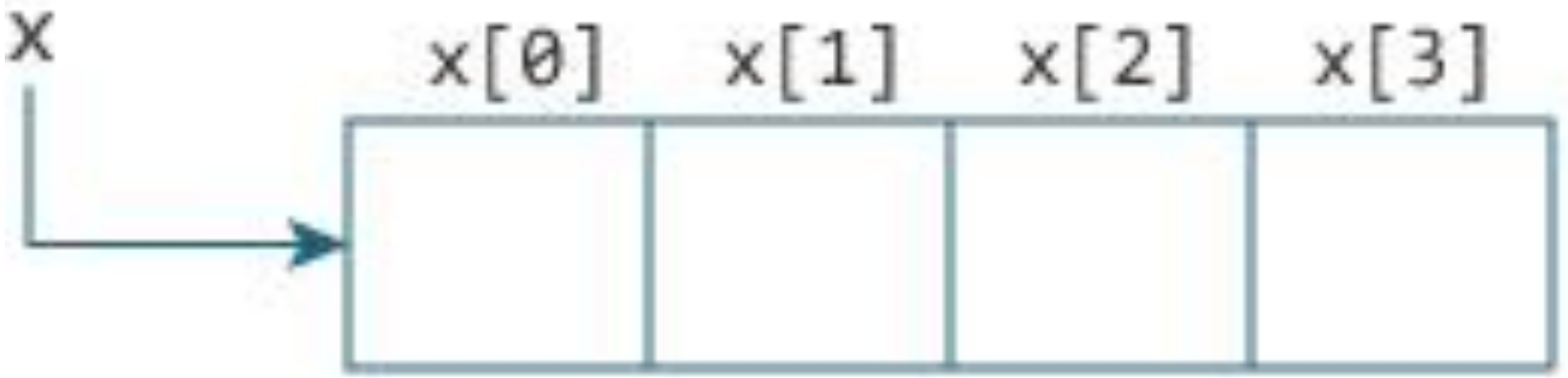
int a[10];

x = &a[0];

x = x + 2 – what will be added is 2 * (size of the variable)

Will make X point to the next integer after the location it originally points to.

X will be address of a[1];



From the above example, it is clear that &x[0] is equivalent to x. And, x[0] is equivalent to *x.

&x[1] is equivalent to x+1 and x[1] is equivalent to *(x+1)

&x[2] is equivalent to x+2 and x[2] is equivalent to *(x+2)

...

&x[i] is equivalent to x+i and x[i] is equivalent to *(x+i)

Example

```
#include <stdio.h>
int main() {
  int i, x[6], sum = 0;
  printf("Enter 6 numbers: ");
  for(i = 0; i < 6; ++i) {
  // Equivalent to scanf("%d", &x[i]);
     scanf("%d", x+i);
  // Equivalent to sum += x[i]
     sum += *(x+i);
  }
  printf("Sum = %d", sum);
  return 0;
}
```

Execution

Enter 6 numbers: 2

3

4

4

12

4

Sum = 29

```
#include <stdio.h>
int main() {
  int x[5] = {1, 2, 3, 4, 5};
  int* ptr;

  // ptr is assigned the address of the third element
  ptr = &x[2];

  printf("*ptr = %d \n", *ptr);   // 3
  printf("*(ptr+1) = %d \n", *(ptr+1)); // 4
  printf("*(ptr-1) = %d", *(ptr-1));  // 2

  return 0;
}
```

*ptr = 3
*(ptr+1) = 4
*(ptr-1) = 2

&x[2] is the address of the third element. It is assigned to the ptr pointer. Hence, 3 was displayed when we printed *ptr.

printing *(ptr+1) gives us the fourth element. Similarly, printing *(ptr-1) gives us the second element.

Write a program to read the values for an array and print the values using points to integers.

Use the same thing for char array and float array