Arrays and Pointers 2

IN 1101 PROGRAMMING FUNDAMENTALS

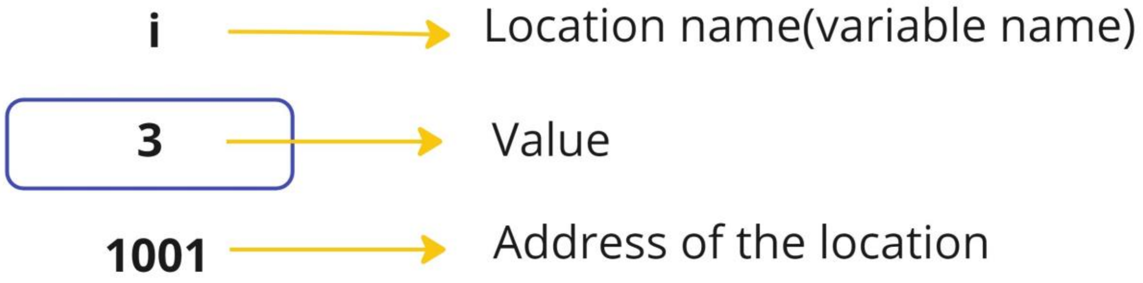
Memory Allocation in C - Recap

Consider the following,

int I = 3;

This tells,

1. Reserve space in memory to hold the integer value.
2. Associate the name ‘i’ with the memory location.
3. Store the value 3 at the location



i → Location name(variable name)

3 → Value

1001 → Address of the location

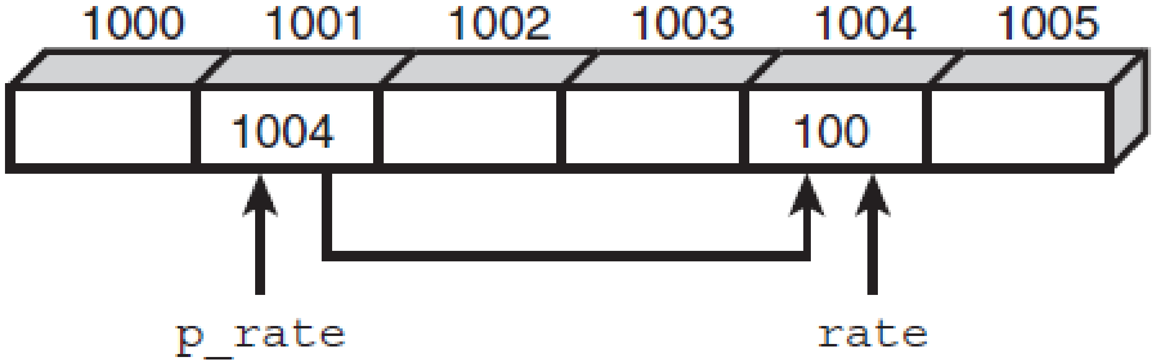
Pointers

❑ If you know a variable’s address, you can create a second variable to store the address of the first.

E.g. : - Create a variable ‘rate’ contains a value of 100 in the address of 1004.

- The address of the ‘rate’ can be stored in another variable called ‘p_rate’.

- Now, p_rate indicates the location where rate is stored in the memory.



Pointers Cont.

- A pointer is a variable that contains the address of another variable.
- Declaring Pointers,

typename *ptrname;

- typename - indicates the type of the variable that pointer points to.

- (*) – indirection operator, indicates that *ptrname* is a pointer to type *typename*(not a variable of type *typename*).

```
e.g. : int  *p_rate;
       char *ch1;
       float *value, percent;
```

- Pointer variable names follow the same rules as other variables and must be unique.

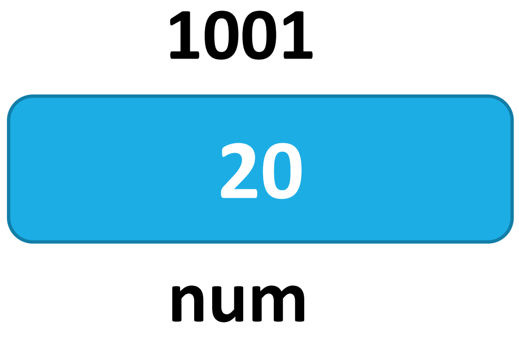
What is the Output here?

```c
#include <stdio.h>
 int num = 20;
 int *ptr;

 int main(void)
 {
    ptr = &num;  // Initialize ptr to point to num
    printf("\nDirect access, var = %d", num);
    printf("\nIndirect access, var = %d", *ptr);

    printf("\n\nThe address of var = %d", &num);
    printf("\nThe address of var = %d\n", ptr);

    return 0;
 }
```



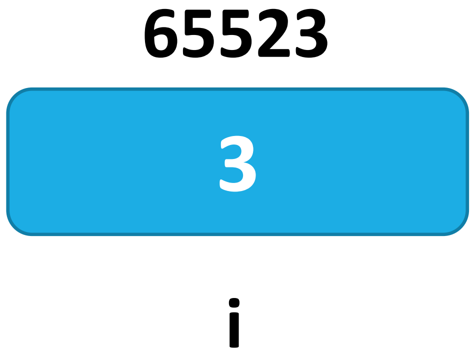
1001

20

num

What is the output here?

```
# include <stdio.h>
int main( )
{
      int i = 3 ;
      printf ( "Address of i = %u\n", &i ) ;
      printf ( "Value of i = %d\n", i ) ;
      printf ( "Value of i = %d\n", *( &i ) ) ;
      return 0 ;
}
```



65523

3

i

Pointers and Arrays

- ❑ When you use array subscript notation, you are really using pointers.
- ❑ Array name without brackets is a pointer to the array’s first element.

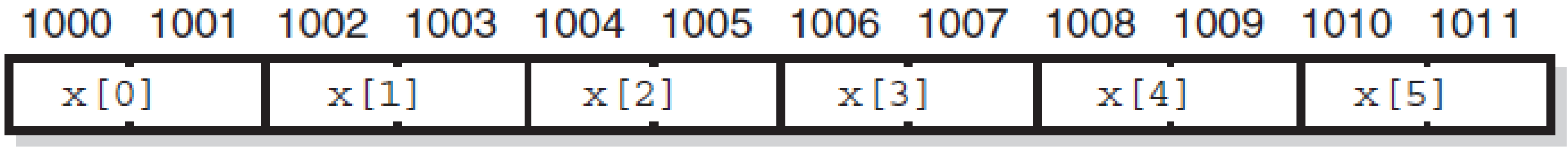
  e.g.: declare an array arr[] , the arr will give the address of the first array element.

  In C, arr == &arr[0] → True
- ❑ Elements of an array are stored in sequential memory locations with the first element in the lowest address.
- ❑ The pointers point to the first element of the array.
- ❑ To access the next element, the pointer must increment by an amount equal to the size of the data type stored in the array.

Incrementing/Decrementing Pointers

- When you increment/decrement a pointer, you are increasing/decreasing its value.
- For example, when you increment a pointer by 1, pointer arithmetic automatically increases the pointer’s value so that it points to the next array element.
- C knows the data type that the pointer points to (from the pointer declaration) and increases the address stored in the pointer by the size of the data type.

Int x[6];



```
# include <stdio.h>
int main( )
{
        int num[ ] = { 24, 34, 12, 44, 56, 17 } ;
        int i ;
        for ( i = 0 ; i <= 5 ; i++ )
        {
                printf ( "address = %u ", &num[ i ] ) ;
                printf ( "element = %d\n", num[ i ] ) ;
        }
        return 0 ;
}
```

Output:

```
# include <stdio.h>
int main( )
{
    int num[ ] = { 24, 34, 12, 44, 56, 17 } ;
    int i, *j ;
    j = &num[ 0 ] ; /* assign address of zeroth element */
    for ( i = 0 ; i <= 5 ; i++ )
    {
        printf ( "address = %u ", j ) ;
        printf ( "element = %d\n", *j ) ;
        j++ ; /* increment pointer to point to next location */
    }
    return 0 ;
}
```

Output:

```
# include <stdio.h>
int main( )
{
    int num[ ] = { 24, 34, 12, 44, 56, 17 } ;
    int i ;
    for ( i = 0 ; i <= 5 ; i++ )
    {
        printf ( "address = %u ", &num[ i ] ) ;
        printf ( "element = %d\n", num[ i ] ) ;
    }
    return 0 ;
}
```

Output:

```
address = 1988098272 element = 24
address = 1988098276 element = 34
address = 1988098280 element = 12
address = 1988098284 element = 44
address = 1988098288 element = 56
address = 1988098292 element = 17
```

```
# include <stdio.h>
int main( )
{
  int num[ ] = { 24, 34, 12, 44, 56, 17 } ;
  int i, *j ;
  j = &num[ 0 ] ; /* assign address of zeroth element */
  for ( i = 0 ; i <= 5 ; i++ )
  {
    printf ( "address = %u ", j ) ;
    printf ( "element = %d\n", *j ) ;
    j++ ; /* increment pointer to point to next location */
  }
  return 0 ;
}
```

Output:

```
address = 1988098272 element = 24
address = 1988098276 element = 34
address = 1988098280 element = 12
address = 1988098284 element = 44
address = 1988098288 element = 56
address = 1988098292 element = 17
```

Array Subscript Notation and Pointers

- If array[] is a declared array, the expression *array is the array’s first element, *(array + 1) is the array’s second element, and so on.

*(array) == array[0]
*(array + 1) == array[1]
*(array + 2) == array[2]
…
*(array + *n*) == array[*n*]

Exercise

Write a program using pointers to find the smallest number in an array of 25 integers.

Multidimensional Arrays

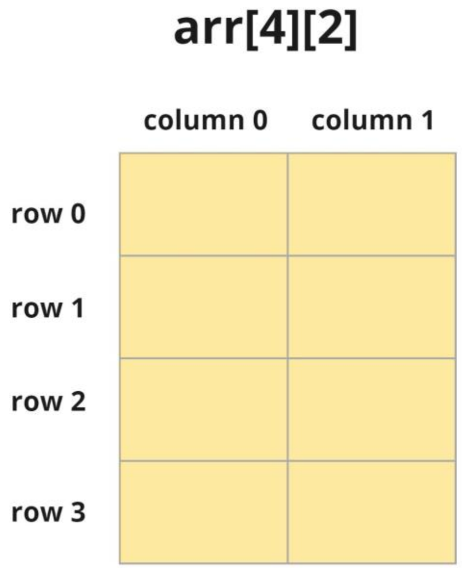
IN 1101 PROGRAMMING FUNDAMENTALS

Two-Dimensional Arrays

- It is also possible for arrays to have two or more dimensions.
- The two-dimensional array is also called a matrix.
- Two- dimensional array is nothing but a collection of a number of one- dimensional arrays placed one below the other.

E.g. : arr[4] [2]

Conceptual map



Initializing a 2-D Array

E.g. :

```
int stud[ 4 ][ 2 ] = {
                        { 1234, 56 },
                        { 1212, 33 },
                        { 1434, 80 },
                        { 1312, 78 }
                        } ;
```

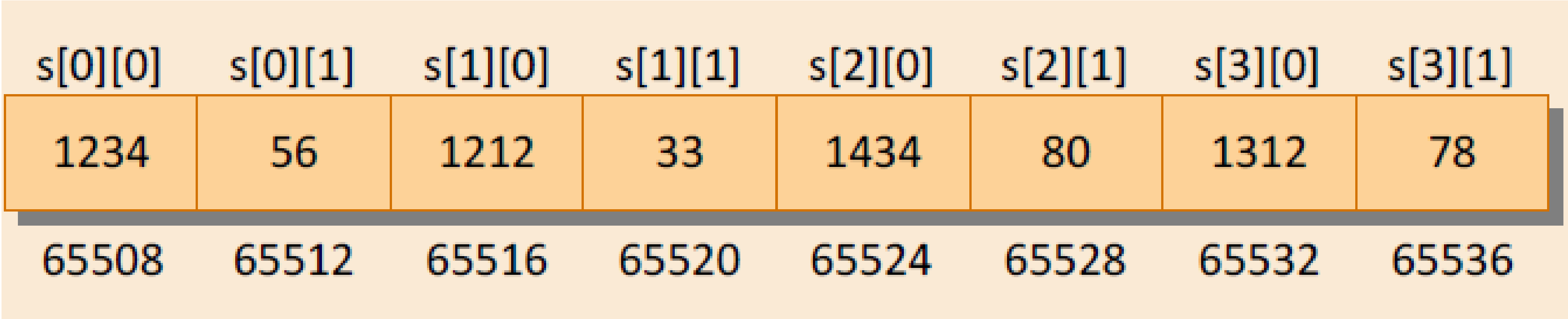
Or

```
int stud[ 4 ][ 2 ] = { 1234, 56, 1212, 33, 1434, 80, 1312, 78 } ;
```

❑ It is important to remember that, while initializing a 2-D array, it is necessary to mention the second (column) dimension, whereas the first dimension (row) is optional.

Memory Map of 2-D Array

- In memory, whether it is a one-dimensional or a two-dimensional array, the array elements are stored in one continuous chain.



s[0][0]	s[0][1]	s[1][0]	s[1][1]	s[2][0]	s[2][1]	s[3][0]	s[3][1]
1234	56	1212	33	1434	80	1312	78
65508	65512	65516	65520	65524	65528	65532	65536

Exercise

1. How will you initialize a three-dimensional array threed[3][2][3]?
2. Write a program to pick up the largest number from any 5 row by 5 column matrix.
3. Write a C program to find the frequency of even numbers in a 2-D matrix(Define your own matrix).

Questions?