COMSATS University
Islamabad
(Lahore Campus)

CSC103-Programming Fundamentals

MS. MAHWISH WAQAS

MAWISH.WAQAS@CUILAHORE.EDU.PK

INTRODUCTION TO POINTERS

Outline

- Background and introduction to pointers
  - Difference with ordinary variables
- Declaring a pointer variable
  - General syntax
  - Example
- Initializing a pointer variable
  - The addressof operator (&)
  - Memory map of an initialized variable
- Indirect reference through pointers
  - The indirection operator (*)
  - 3 uses of * operator
- Valid/Invalid Operations with pointers

Background

- We know that functions, which can return a single value back to its caller.
- Often, it is required to return multiple values from a function, e.g.
  - Find the 2 quadratic roots of an equation provided the values of a, b and c. In this case, 2 values should be returned.
- A function can return multiple values by passing *output parameters* (also called pass-by-reference)
  - *Output parameters* are passed by Indirect addressing using pointers.

Introduction to Pointers

- A pointer or pointer variable is a memory cell that stores the address of a data item.
- Compare with ordinary scalar (non-pointer) variables:
  - Scalar variable stores the value of a data item.
  - Pointer stores the address of a data item.
- The address of a data item is represented by ‘&’ operator.
- Like ordinary variables, a pointer variable should be declared before using it in the program.

Declaring Pointers

- General syntax of *declaring a pointer*:
  - *type* *ptr_name;
  - The operator * is *pointer declaration operator*.
  - ptr_name is a pointer variable for storing address of a variable with datatype=*type.*
  - The value of the pointer variable ptr_name is a memory address.
- Examples
  - int *ptr_int; // stores the address of an integer variable.
  - char *ptr_char; // stores the address of an char variable.
  - float*   ptr_float; // stores the address of an float variable

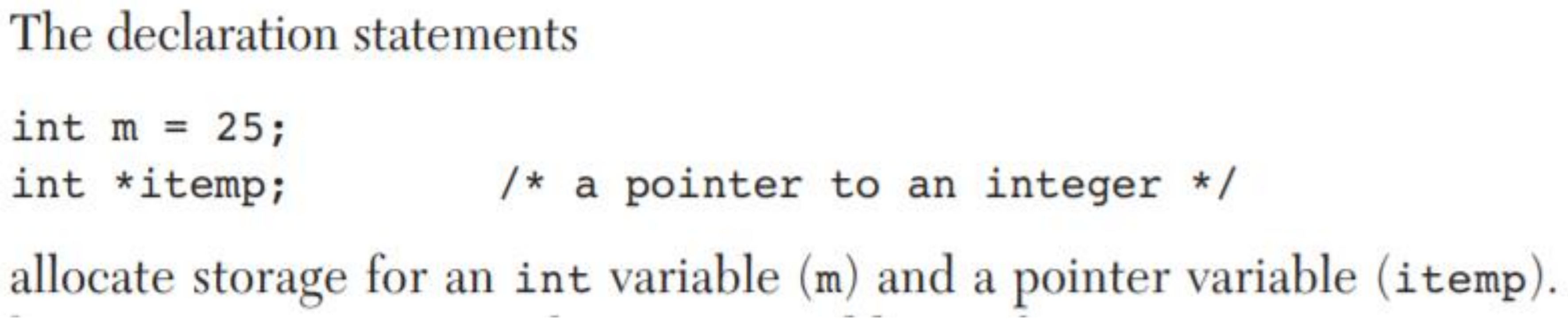
The Addressof (&) operator

- The address of a data item (variable) can be obtained using the addressof operator ‘&’.
  - Also called the ampersand operator.
- Simply place the addressof operator in front of the data item’s name.
  - E.g. &intVar represents the address of an integer variable intVar.
- Try printing this value:
  - `cout << &intVar;`

Initializing Pointer variable

We can initialize a pointer by storing in it the address of another variable.

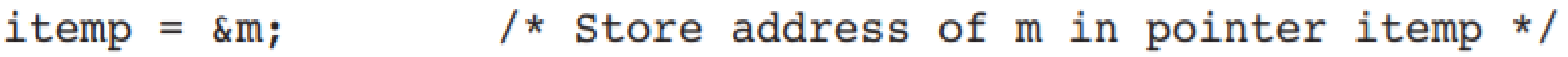
Example:



The declaration statements

```
int m = 25;
int *itemp;          /* a pointer to an integer */
```

allocate storage for an int variable (m) and a pointer variable (itemp).



```
itemp = &m;          /* Store address of m in pointer itemp */
```

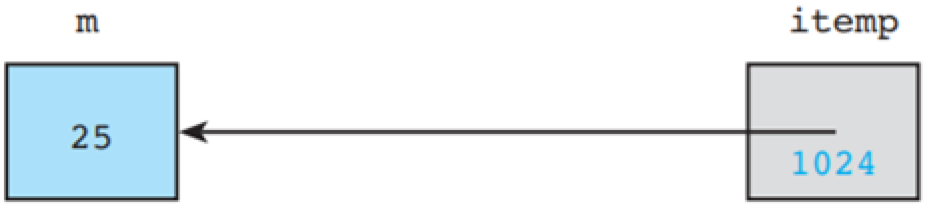
- Note that itemp can only store address of an integer variable, as it a “pointer to integer”.

Initializing Pointer variable (Contd.)

The following statement is referred as: The pointer itemp points to integer m.

```
itemp = &m;
```

After this statement, the memory map can be visualized as follows, where arrow points to the memory cell, whose address is stored in itemp:

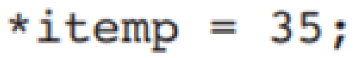


Here, it is assumed that the address of m in memory is 1024.

- The above statement stores 1024 in itemp, i.e. itemp now indirectly refers to the variable m.

Indirect reference – the indirection operator

- After initializing a pointer, we can indirectly access/change the value at stored address, as shown in the following example.



```
*itemp = 35;
```

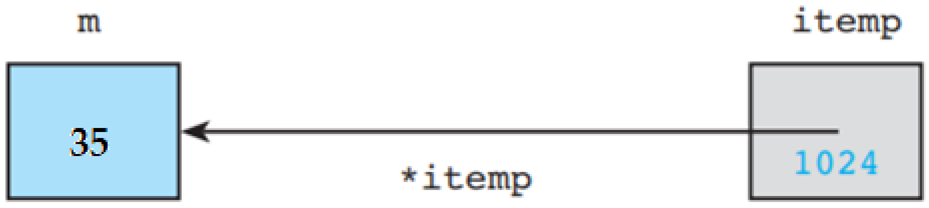
  - The * operator in this case is called the indirection operator or dereference operator.
  - This operation of using the * operator to access the contents of a memory cell is called *dereferring* / *indirecting* a pointer variable.

- The Indirection operator is used to access the contents of a memory cell through a pointer variable that stores its address.
  - The above statement writes to the memory cell represented by m (i.e. 1024) *indirectly* using pointer itemp (recall itemp stores the address of m).

Indirect reference – the indirection operator (Contd.)

```
*itemp = 35;
```

- The effect of above statement can be visualized as follows:



- The indirection operator can be read as access the memory cell by "*following the pointer*".
  - Once you *follow the pointer*, you reach the memory address of an integer, therefore, the type of *itemp is integer.

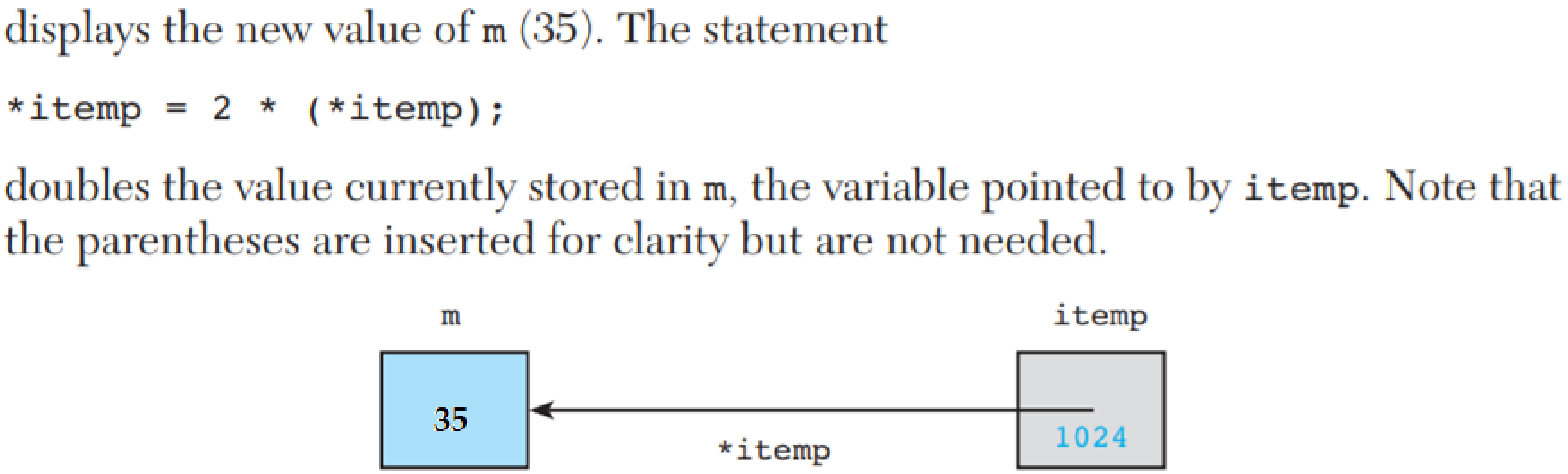
Reference	Cell referenced	Cell Type (Value)
itemp	gray shaded cell	pointer (1024)
*itemp	cell in color	int (25)

Indirect reference – the indirection operator (Contd.)

As the type of *itemp is integer, any operation valid with integers can also be performed with *itemp.

A couple of examples are given below.

```
cout << *itemp;
```



displays the new value of `m` (35). The statement

```
*itemp = 2 * (*itemp);
```

doubles the value currently stored in `m`, the variable pointed to by `itemp`. Note that the parentheses are inserted for clarity but are not needed.

m
35
*itemp
itemp
1024

3 Uses of the * Operator

Note that we have seen 3 uses of the * operator:

Operator name	Type	Purpose
Multiplication operator	Binary	Multiply its operands
Pointer declaration operator	Unary	Declare a pointer variable; only used once with the pointer while declaring it.
Indirection / dereference operator	Unary	Follow the pointer to access the pointed memory cell; always used once the pointer has been declared.

Note the difference clearly and never confuse them together.

The NULL Pointer

- Note that dereferring an uninitialized pointer results in a runtime error. E.g.
  - `int *ptr; // an uninitialized pointer`
  - `cout << *ptr << endl; // results in a runtime error.`
- It is a good practice to set uninitialized pointer variables to a special value, i.e. NULL, in order to indicate that the pointer does NOT point to any location in memory.
  - `int *ptr=NULL;`
  - **Note NULL is case-sensitive.**
- A pointer which stores NULL is called a *NULL pointer*.

The NULL Pointer (Contd.)

- We can check for a NULL pointer before accessing any pointer variable.
  - By doing so, we can perform error handling in pointer related code e.g. dereference pointer variable only if it’s NOT NULL.
- Example:

```
int *pInt = NULL;
if(pInt != NULL) /*We could use if(pInt) as well*/
{ /*Some code*/}
else
{ /*Some code*/}
```

Valid operations with Pointers

Not all operations with ordinary scalar variables are valid with pointer variables.

The only valid operations with pointer variables are:

1. *Adding/subtracting an integer* to/from a pointer variable (Increment/decrement is a special case of addition/subtraction)
2. *Subtracting 2 pointer variables* of same type
3. *Compound assignment operators* with integers for addition/subtraction (+=, -=)
4. Pointers comparison using *relational operators* (<, >, <=, >=, ==, !=)

Valid *arithmetic* operations with pointers are NOT performed the same way as ordinary scalar variables.

- The results of *arithmetic* operations depend on the type of pointer. We will discuss it after we discuss array lecture.

Invalid operations with Pointers

- All other operations such as the following are INVALID with pointer variables.
  - Multiplication/division of a pointer with integers/other pointers
  - Addition of 2 pointer variables
  - Compound assignment operators of types other than listed previously (i.e. *=, /=, %=)

Pointer arithmetic

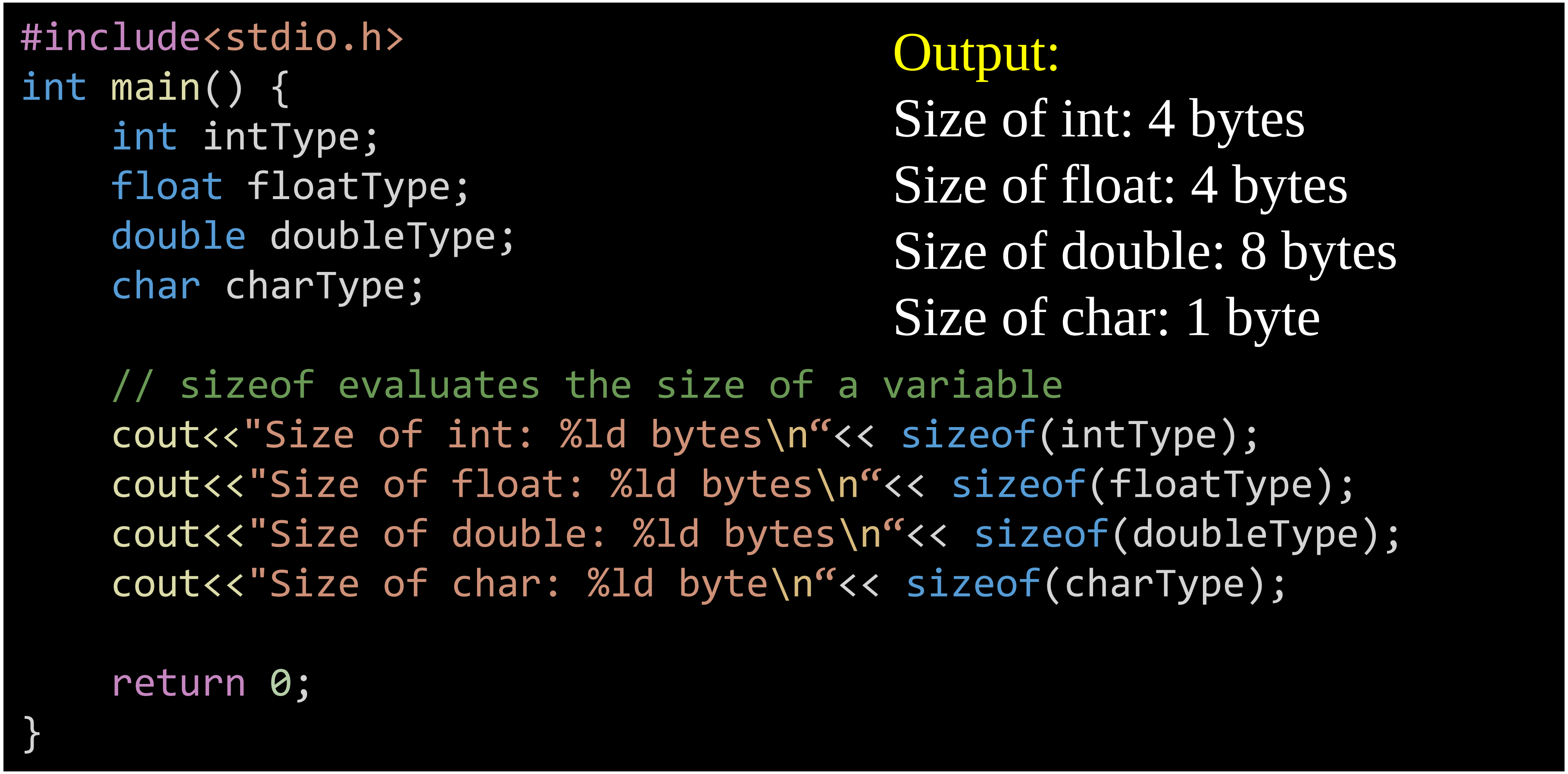
- **Recall from “Introduction to pointers”**

- Valid arithmetic operations with pointer variables are:
  - *Adding/subtracting an integer* to/from a pointer variable
  - *Subtracting 2 pointer variables* of same type

- Valid *arithmetic* operations with pointers are NOT performed the same way as ordinary variables.
  - The results of *arithmetic* operations depend on the type of the pointer (pointer to integer, pointer to float etc.).
    - The type determines the size of variable in memory, which *actually* affects the result of operation.

The sizeof Operator

- The sizeof operator provides the size of a type/variable in memory.
  - Can be used like a function call with 1 argument, i.e. sizeof(<type>); OR sizeof(<var_name>);
- Examples:
  - Using type name as argument: e.g. sizeof(int); OR sizeof(double);
  - Using variable name as argument: e.g. sizeof(x);



```
#include<stdio.h>
int main() {
    int intType;
    float floatType;
    double doubleType;
    char charType;

    // sizeof evaluates the size of a variable
    cout<<"Size of int: %ld bytes\n“<< sizeof(intType);
    cout<<"Size of float: %ld bytes\n“<< sizeof(floatType);
    cout<<"Size of double: %ld bytes\n“<< sizeof(doubleType);
    cout<<"Size of char: %ld byte\n“<< sizeof(charType);

    return 0;
}
```

Output:

```
Size of int: 4 bytes
Size of float: 4 bytes
Size of double: 8 bytes
Size of char: 1 byte
```

Note that the size of different types may vary on different compilers.

Valid operations with Pointers (Contd.)

Consider the following code:

```
int m=20; // Assume each integer takes 4 bytes in memory
int  n=30; // Assume m and n are placed consecutively in memory.
int *ptr=&m; // Assume address of m = 1024
int *ptr2=&n; // Assume address of n = 1028
// ASSUME x=1 in the following examples.
```

Operation	Operands	Expression	Result	Comment
Addition	Pointer (ptr) and integer (x)	ptr+x = ptr+x*sizeof(int)	1024+1*sizeof(int) =1024+1*4 =1028	Adding an integer ‘x’ means jump ‘x’ integers forward in memory.
Subtraction	Pointer and integer	ptr-x = ptr-x*sizeof(int)	1024-1*sizeof(int) =1024-1*4 =1020	Subtracting an integer ‘x’ means jump ‘x’ integers backward in memory.
Subtraction	Pointer (ptr) and pointer (ptr2)	Ptr2-ptr =(ptr2-ptr)/sizeof(int)	(1028-1024) / sizeof(int) =(1028-1024)/4 =1	Subtracting 2 pointers means how many integers apart are the 2 pointers in memory.

Common Programming Error

It is an invalid operation to add/subtract a double/float value to/from a pointer.

Valid operations with Pointers (Contd.)

- The operations in above example with integer pointer can be generalized to pointers to any type.
- For example, in case of pointer to double:
  - Adding an integer ‘x’ means jump ‘x’ double forward in memory.
  - Subtracting an integer ‘x’ means jump ‘x’ double backward in memory.
  - Subtracting 2 pointers to double means how many double apart are the 2 pointers in memory.

Valid operations with Pointers (Contd.)

- In general, if ptr and ptr2 are pointers to <type>, x is an integer, then:
  - ptr+x Evaluates as ptr+x*sizeof(<type>)
  - Ptr-x Evaluates as ptr-x*sizeof(<type>)
  - Ptr2-ptr Evaluates as (ptr2-ptr)/sizeof(<type>)