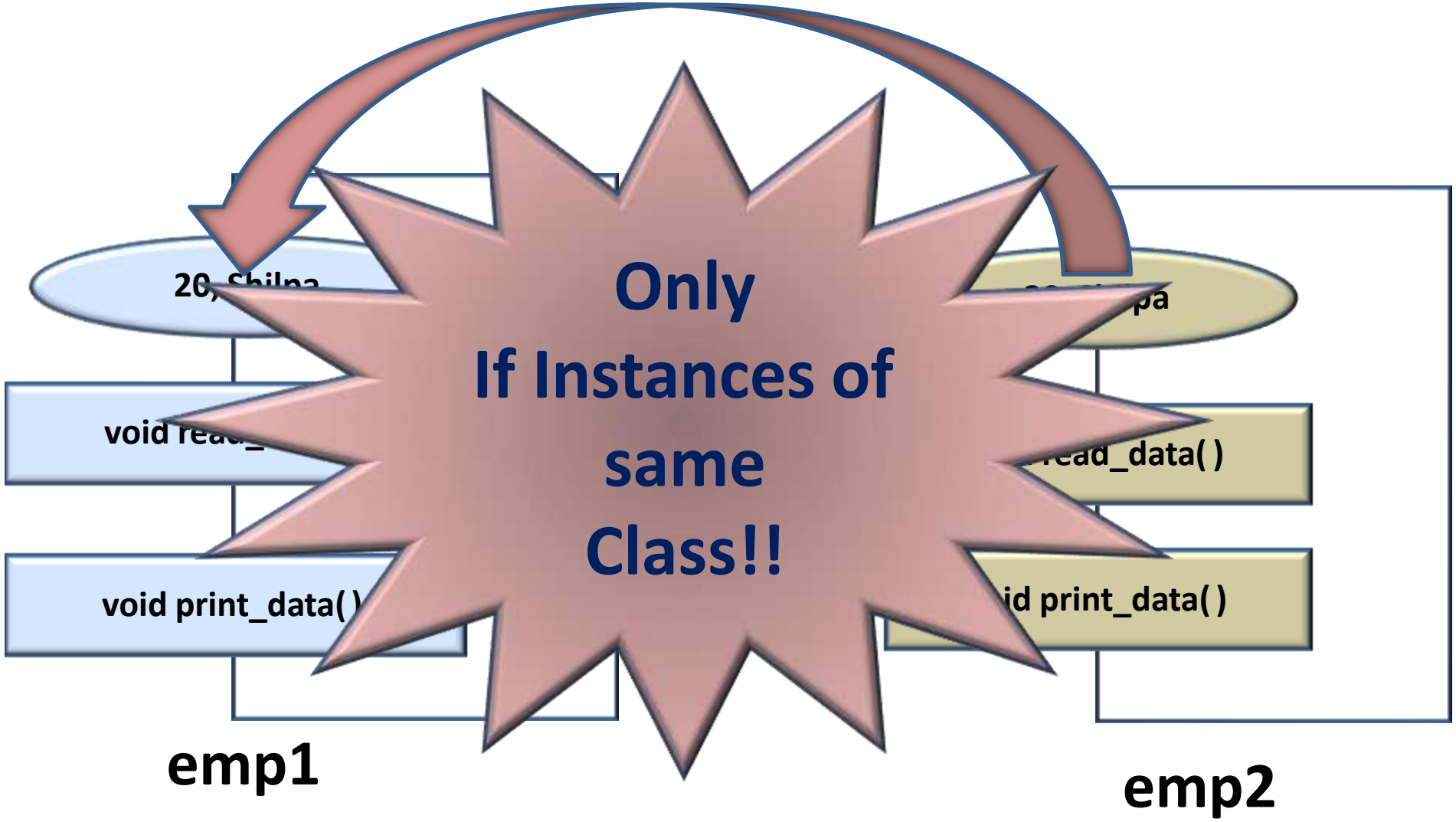
CSC127 Objects and Classes

Passing Objects as Arguments

- Objects are passed to functions through the use of the standard **call-by-value** mechanism.

- Means that a **copy of an object** is **made** when it is passed to a function.

Object Assignment



Passing Objects as Arguments

```
class complex
{
    ......
    ......
  void Add(int x, complex c);
    ......
    ......
};
```

```
void main()
{

    complex  obj,  s1;
        ......
        ......
        ......
  obj.Add(6,   s1) ;
        ......
        ......
}
```

Returning Objects

- A function may return an object to the caller.

```
class complex
{
    ......
    ......
  complex Add(int x, complex c);
    ......
    ......
};
```

```
void main()
{

    complex  obj,  s1;
        ......
        ......
        ......
  obj=obj.Add(6,   s1) ;
        ......
        ......
}
```

Friend Functions

- “Friend function is a **non-member function** which can **access** the **private members** of a class”.

- To declare a **friend function,** its **prototype** should be included within the class, preceding it with the keyword **friend.**

Friend Functions

Example:

```
class myclass
 {
    int a, b;
    public:
    friend int sum(myclass x);
    void set_val(int i, int j);
};
```

Syntax:

```
class class_name
{
//class definition
public:
friend rdt fun_name(formal parameters);
};
```

Friend Functions

Advantages…?

- When we overload operators.
- When we create I/O overloaded functions.
- When two or more classes members are interrelated and to carry out the communication to other parts of the program.

Friend Classes

"A class can be a friend of another class, allowing access to the **protected** and **private** members of the class in which is defined."

Friend class and all of its member functions have access to the private members defined within the that class.

Friend Classes

```
class Aclass
{
public :
        :
friend class Bclass;
private :
int Avar;
};
```

Friend class Declaration

- class Bclass
- {
- public :
    - :
- void fn1(Aclass ac)
- {
  - Bvar = ac. Avar;
- }
  - private :
    int Bvar;
- };

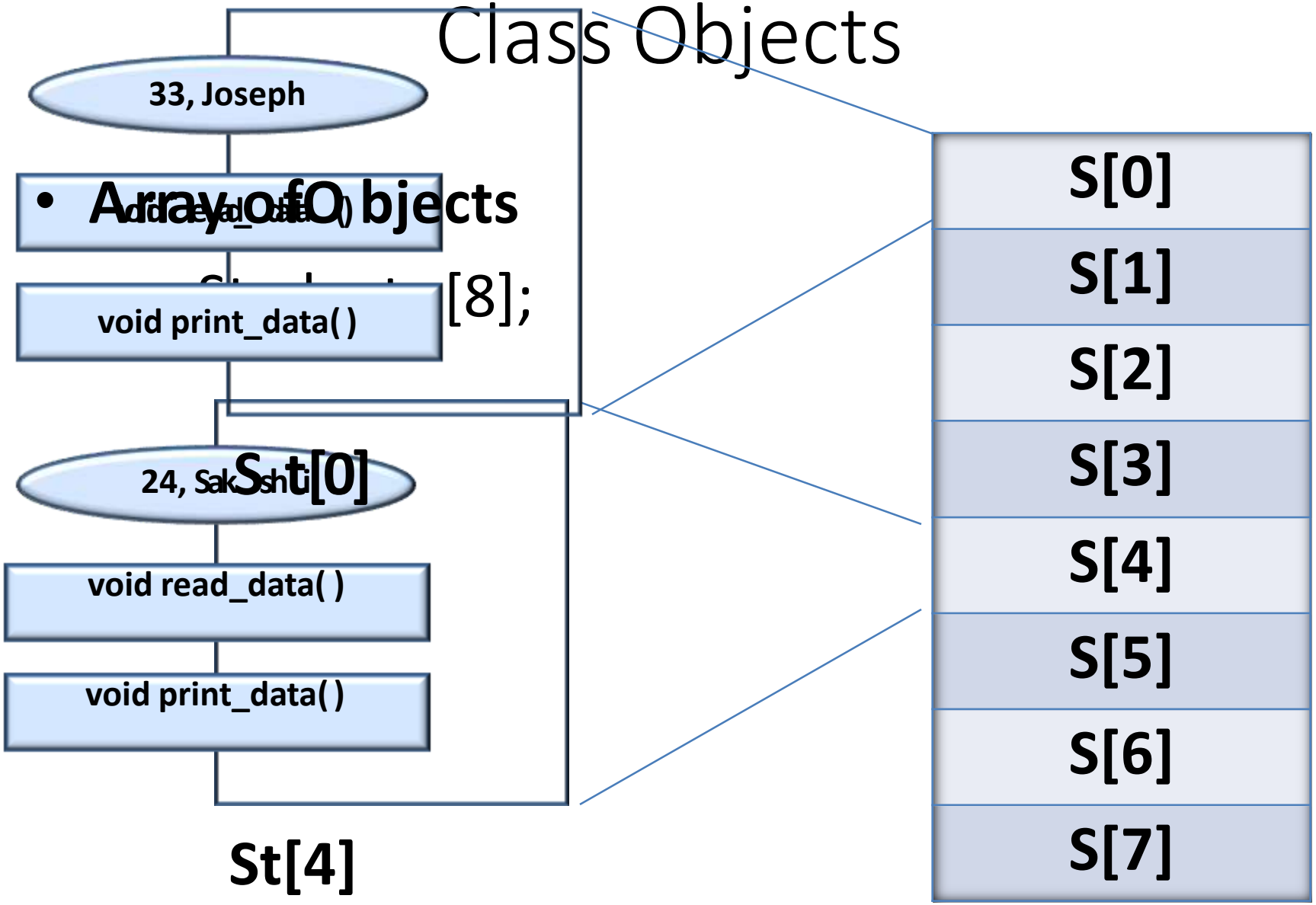
Arrays of Objects

- **Several** objects of the **same class** can be declared as an array and used just like an array of any other data type.

- The **syntax** for declaring and using an object array is **exactly** the **same** as it is for any other type of array.

Class Objects

- Array of Objects



Dynamic Objects

“ Dynamic objects are objects that are **created** / **Instantiated** at the **run time** by the class”.

- They are **Live Objects**, initialized with necessary data at run time.

- Its life time is explicitly managed by the program(should be handled by programmer).

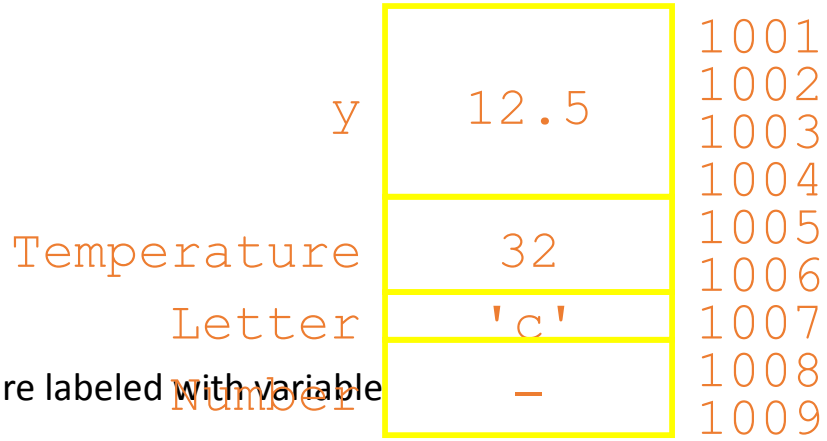
Dynamic Objects

- The **new** operator is used to create dynamic objects.

- The **delete** operator is used to release the memory allocated to the dynamic objects.

NOTE:
C++ does not have **Default Garbage Collector.**

Pointer Variables



- *variable* is a named memory location
- variable *value* is data stored in variable
  - variable always has a value
- compiler removes variable name and assigns memory location
  - however, it is convenient to think that memory locations are labeled with variable names

  - Char Letter;
  - Char *PtrLetter;
  - Pointer to Letter === same as 1007
  - Letter =‘c’;
  - &Letter ==== 1007 which is the pointer to Letter;
  - PtrLetter = &Letter;
  - *PtrLetter != &Letter; {pointer indirection}!!

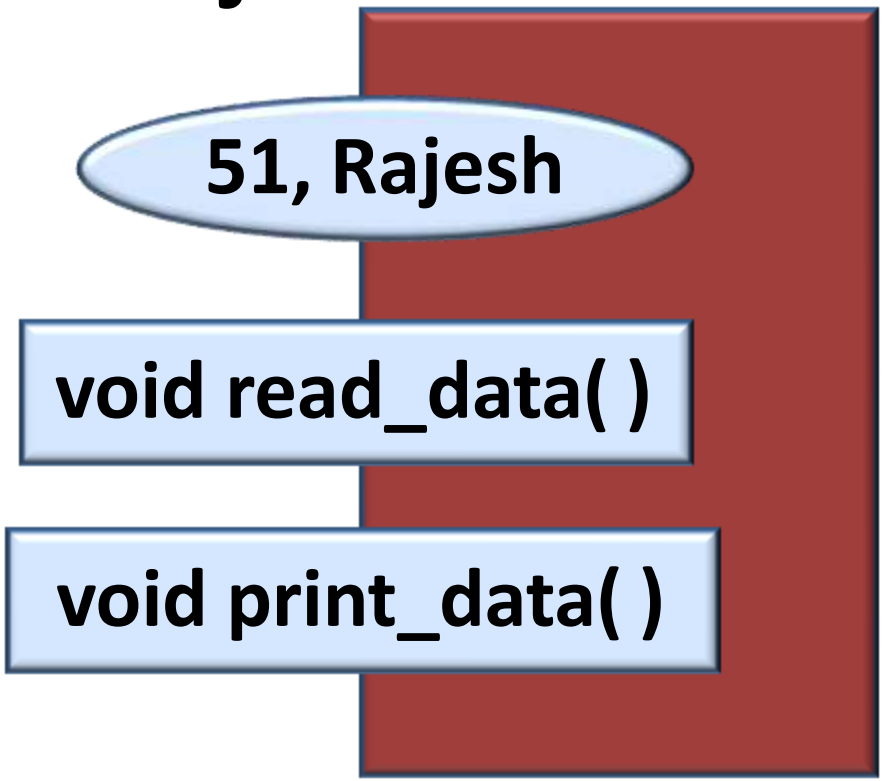
Pointers to Objects

```
student   st;

student  *ptr;

ptr   =   & st;

*ptr = str;
```

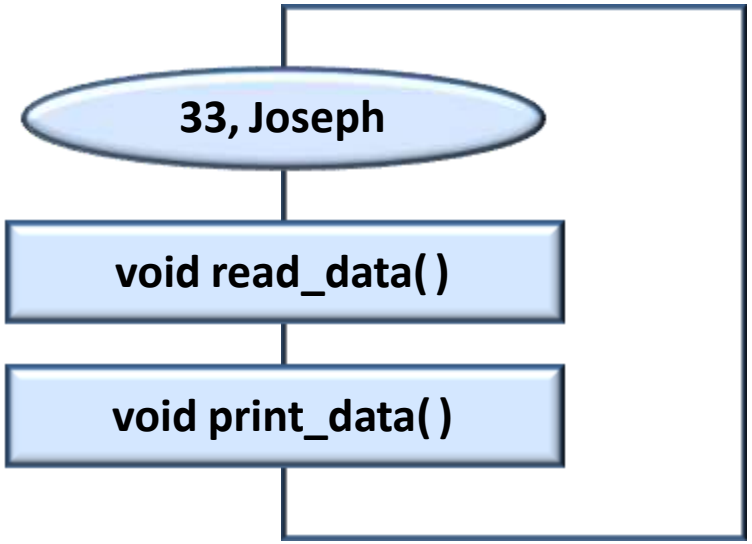


st

2FCD54

Pointers to Objects

"**Pointers** can be defined to **hold** the **address** of an **object**, which is created statically or dynamically".



st

2FCDA4

Statically created object:

```
student     *stp;
stp  =  &st;
```

Dynamically created object:

```
student   *stp;
stp    =    new student;
```

Pointers to Objects

- **Accessing Members of objects:**

  Syntax:

  ptr_ob j → member_name;

  ptr_obj → memberfunction_name( );

  Example:

  stp → st_name;

  stp → read_data ( );

The *this* Pointer

"The **this** pointer points to the object that invoked the function".

- When a member function is called **with** an **object**, it is **automatically passed** an implicit argument that is a **pointer** to the invoking object (that is, the object on which the function is called).

The *this* Pointer

- **Accessing Members of objects:**

  Syntax:

  obj . memberfunction_name( );

Example:

st . read_data ( );

this pointer points to **st** object