

Reference I

- reference is just an *alias to an existing object*

- reference declaration operator: &

```
int level = 1;  
int &debug_level_ref = level;
```

- this & is not same as the address operator because its only used in the declaration

- note a reference has to be initialized at the time of declaration and should refer to an existing entity of the same type (here level)

- usage

```
cout << "debug_level_ref: " << debug_level_ref << endl;
```

- note unlike pointers no *s necessary to get the value of the reference

Reference II

- references and pointers:

```
int
main (int argc, char **argv)
{
    int level = 1;
    int &debug_level_ref = level;
    int *debug_level_ptr = &level;

    cout << "debug_level_ptr: " << debug_level_ptr << endl
         << "*debug_level_ptr: " << *debug_level_ptr << endl
         << "debug_level_ref: " << debug_level_ref << endl
         << "&debug_level_ref: " << &debug_level_ref << endl;
}
```

- what is the relation between &level, &debug_level_ref and debug_level_ptr?

Reference III

- what won't work:

```
/*  
 * The following two lines won't work because reference has to  
 * be initialized at the time of declaration  
 */  
int &debug_level_ref;  
debug_level_ref = level;
```

- so always initialize references
- because references always need to be initialized there could be no NULL references
- references cannot be re-initialized, i.e. throughout its lifetime it will serve as an alias to the object it was first initialized with

Reference IV

- consider the following function:

```
int
test (int val, int *ptr, int &ref)
{
    ++val;
    ++(*ptr);
    ++ref;
}
```

- what should the following do?

```
level = 3;
cout << "*debug_level_ptr: " << *debug_level_ptr << endl
     << "debug_level_ref: " << debug_level_ref << endl;

test(level, debug_level_ptr, debug_level_ref);
cout << "level: " << level << endl
     << "*debug_level_ptr: " << *debug_level_ptr << endl
     << "debug_level_ref: " << debug_level_ref << endl;

test(level, &level, level);
cout << "level: " << level << endl;
```

Reference V

- difference with C: in C++, pass by reference has a distinct meaning
- although there are big debates on this issue, here is some general advice on when to use what while calling/writing a function:
- when to use references?
 - use references when you can, and pointers when you have to
 - references are usually preferred over pointers whenever you don't need re-initialization or "reseating"
- when to use pointers?
 - use a pointer when a function's return value needs to be NULL to indicate "failure" because you cannot return a NULL reference

Reference VI

- some more advice:
 - pass by value those variables which carry small amount of data, i.e., copying isn't time consuming
 - pass by const reference those variables which carry large amount of data e.g. a huge structure whose value *may not* be changed by the called function
 - “pass by value” using pointers those variables which carry large amount of data e.g. a huge structure / array whose value *may* be changed by the called function

Code Files

prog6.C