

Garbage Collection

- normally in C++ the default mechanism for managing dynamic memory is to use `new` and `delete`
- which are very similar to `malloc` and `free` found in C
- we note that in C++ we also need to provide copy and assignment operators
 - these also must copy dynamic data, often invoking `new` and `delete`

Garbage Collection

- there is a problem in handling the dynamic data, in particular returning dynamic data which is no longer required back to the free pool
- C++ manages this by its rule of three discussed in earlier lectures
- this works well
 - every time an object goes out of scope it is deleted
- however it can be costly, sometimes in time critical applications it might be better to delay the deallocation until later
 - consider real-time games
 - should be possible to create a thread to run the deallocation in parallel with the event loop

Example code

- ```
sfract v;
sfract s;
sfract radians;
int i;
sfract two_pi = 2 * pi ();

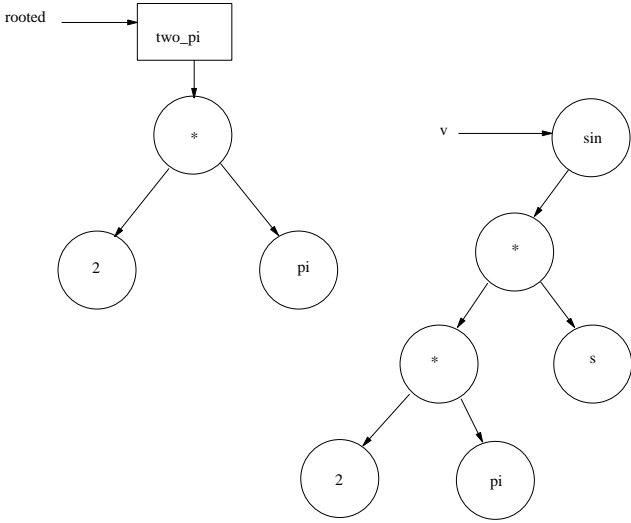
two_pi.root ();

for (i = 0; i<360; i += 15)
{
 s = sfract (i, 360);
 radians = s * two_pi;
 v = sin (radians);
 std::cout << "sin (" << i << " degrees) can be expressed as "
 v = v();
 std::cout << " and also " << v << "\n";
 sfract_garbage_collect ();
}
```

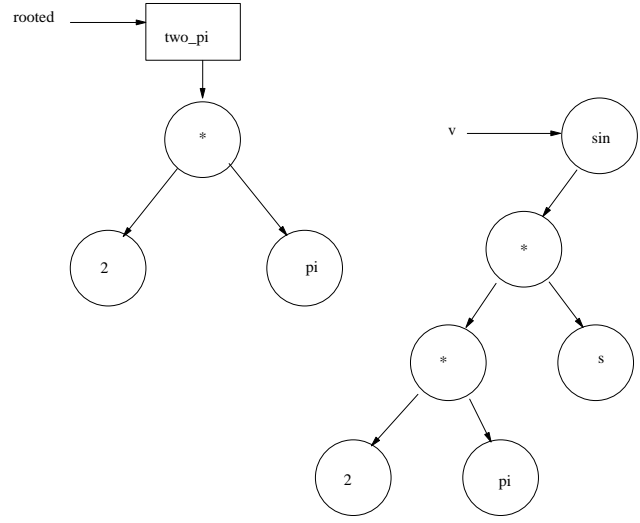
## Garbage collection algorithm

- based on mark and sweep
- the programmer roots critical data structures
  - indicating that these data structures must survive the garbage collection
  - also all rooted dependent data structures must survive the garbage collection
- in the above example the variable `two_pi` is rooted

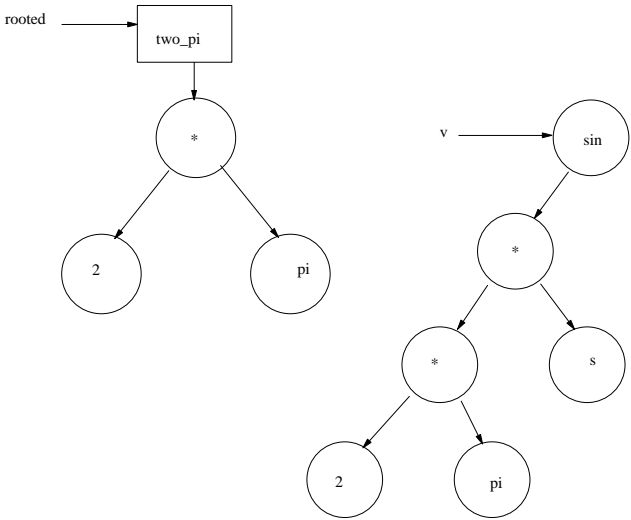
### Action of Garbage collection



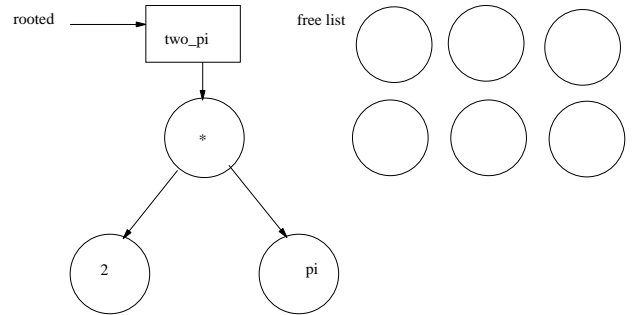
### Mark all as candidates for removal



### Set all root nodes and dependants used



### Sweep any nodes still marked into free list



**gc.h**

examples/c++/fractions/gc.h

```
class gc
{
private:
 entity *rooted;
 entity *allocated;
 int bytes;
 char *desc;
 entity *free_list;
 gc *next;
};
```

**gc.h**

examples/c++/fractions/gc.h

```
public:
 gc (int no_of_bytes, const char *description);
 ~gc ();
 void collect (void);
 void *allocate (entity *e);
 void root (entity *e);
 void unroot (entity *e);
 void *get_data (entity *e);
 entity *get_entity (void *data);
 bool is_rooted (entity *e);
 void mark_allocated (void);

 void stats (void);
 int no_of_allocated (void);
 int no_of_freed (void);
 int no_of_rooted (void);
};
```

**gc.h**

examples/c++/fractions/gc.h

```
/*
 * garbage_collect - pre-condition : none.
 * post-condition: all the garbage col.
 * will attempt to rec.
 * memory.
 */
void garbage_collect (void);

/*
 * allocate - pre-condition: init_garbage has been call,
 * maintain a, bytes, heap.
 * post-condition: entity, e, is filled in an
 * allocated memory is return
 */
void *allocate (unsigned int bytes, entity *e);
```

**gc.h**

examples/c++/fractions/gc.h

```
/*
 * init_garbage - pre-condition : none.
 * post-condition: a garbage collector i
 * to serve calls for by
 * of memory.
 */
gc *init_garbage (unsigned int bytes, const char *descrip

#endif
```

**gc.h**

examples/c++/fractions/gc.h

```

typedef enum {freed = 1, marked = 2, in_use = 4,
 in_error=8, max_state = 16} state;

class entity
{
public:
 void *data;
 state status;
 entity *a_next;
 entity *r_next;
 entity *f_next;

 entity (void);
 ~entity (void);
 entity (const entity &from); // copy
 entity& operator= (const entity &from); // assignment

```

**gc.h**

examples/c++/fractions/gc.h

```

void unmark (void);
void mark (void);
bool is_marked (void);

void free (void);
void unfree (void);
bool is_free (void);

void used (void);
void unused (void);
bool is_used (void);

void do_assert (void);
};

```

**gc.cc**

examples/c++/fractions/gc.cc

```

void *operator new (std::size_t bytes)
{
 entity *e;

 return allocate (bytes, e);
}

```

**gc.cc**

examples/c++/fractions/gc.cc

```

/*
 * allocate - pre-condition: init_garbage has been called to
 * maintain a, bytes, heap.
 * post-condition: entity, e, is filled in and the
 * allocated memory is returned.
 */

void *allocate (unsigned int bytes, entity *e)
{
 if ((bytes == sizeof (entity)) || (bytes == sizeof (gc)))
 return malloc (bytes);

 gc *g = list_of_gc->find_gc (bytes);

 if (g == 0)
 return malloc (bytes); // no garbage collector initialised yet
 else
 return g->allocate (e);
}

```

## Running your code before you have implemented garbage collection

■

```

sin (0 degrees) can be expressed as sin ((0 * (2 * pi))) .
total number of entities 55

sin (15 degrees) can be expressed as sin ((1/24 * (2 * pi))) .
total number of entities 114

sin (30 degrees) can be expressed as sin ((1/12 * (2 * pi))) .
total number of entities 173

sin (45 degrees) can be expressed as sin ((1/8 * (2 * pi))) .
total number of entities 232
...

sin (315 degrees) can be expressed as sin ((7/8 * (2 * pi))) .
total number of entities 1294

sin (330 degrees) can be expressed as sin ((11/12 * (2 * pi))) .
total number of entities 1353

sin (345 degrees) can be expressed as sin ((23/24 * (2 * pi))) .
total number of entities 1412

```

## Running your code once you have implemented garbage collection

■

```

sin (0 degrees) can be expressed as sin ((0 * (2 * pi))) .
total number of entities 109

sin (15 degrees) can be expressed as sin ((1/24 * (2 * pi))) .
total number of entities 145

sin (30 degrees) can be expressed as sin ((1/12 * (2 * pi))) .
total number of entities 145

sin (45 degrees) can be expressed as sin ((1/8 * (2 * pi))) .
total number of entities 145
...

sin (315 degrees) can be expressed as sin ((7/8 * (2 * pi))) .
total number of entities 145

sin (330 degrees) can be expressed as sin ((11/12 * (2 * pi))) .
total number of entities 145

sin (345 degrees) can be expressed as sin ((23/24 * (2 * pi))) .
total number of entities 145

```