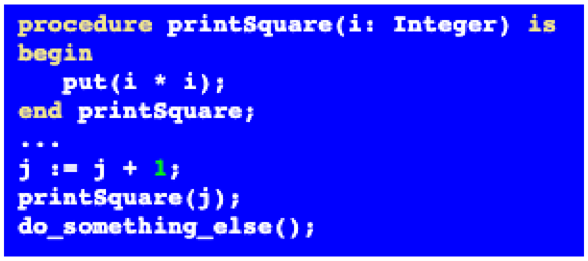
1 Thread API

Vocabularies

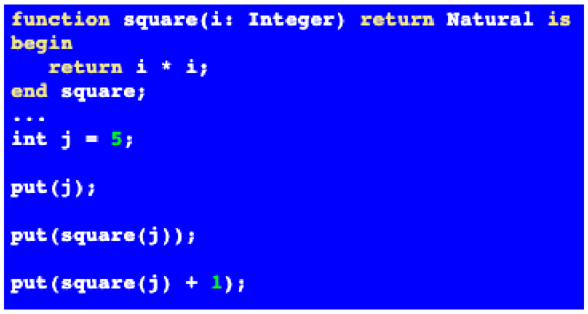
1. **Void Pointer**
   - Is a pointer in C that has no associated data type with it
     - Can hold address of any type and can be typcasted to any type
2. **Procedure Call**
   - Is synonymous to void function in C
   - Is a statement without return value

   Example



```
procedure printSquare(i: Integer) is
begin
   put(i * i);
end printSquare;
...
j := j + 1;
printSquare(j);
do_something_else();
```

3. **Function call**
   - Is synonymous to non-void function in C
   - Is a statement with return value



```
function square(i: Integer) return Natural is
begin
   return i * i;
end square;
...
int j = 5;

put(j);

put(square(j));

put(square(j) + 1);
```

4. **Locks**
   - Is also called **mutex**
   - Is a **synchronization primitive** used to prevent multiple threads from accessing a shared resource at the same time

5. **Critical Section**
   - Is a piece of code that acceses a shared resource, usually a variable or data structure

1.1 Thread Creation

- Is done using `pthread_create`

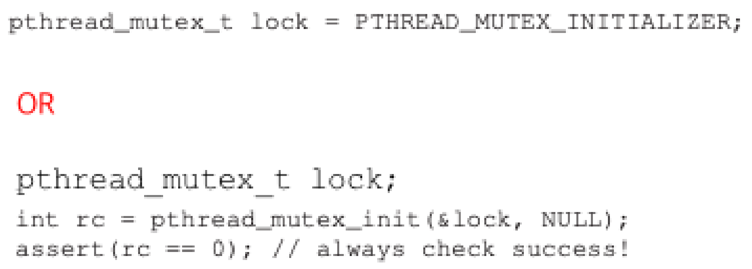
1.2 Thread Completion

- Is done using `pthread_join`
  - Suspends the execution of the calling thread until the target thread (that's in `pthread_join`) terminates

1.3 Locks

- Ensures only one thread can enter **critical section** at a time
- Use
  1. Properly initialize lock

     Example



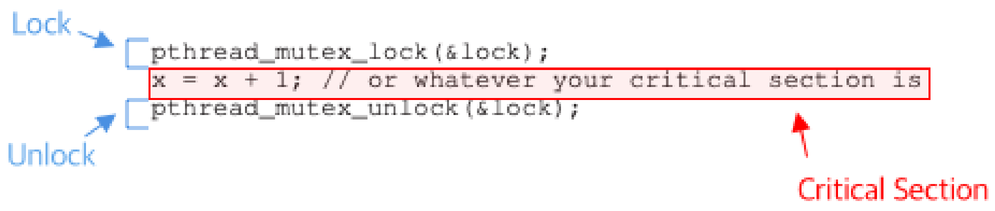
```
pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;
```

OR

```
pthread_mutex_t lock;
int rc = pthread_mutex_init(&lock, NULL);
assert(rc == 0); // always check success!
```

  2. Wrap `lock` and `unlock` around **critical section**

     Example



```
pthread_mutex_lock(&lock);
x = x + 1; // or whatever your critical section is
pthread_mutex_unlock(&lock);
```

3. Destroy lock after use

   Example

   ```
   pthread_mutex_destroy(&lock);
   ```

1.4 Conditional Variable

- Is a queue that threads can put themselves on when it's execution is not desired
  - Queue is FIFO
- Forces thread(s) to wait until another thread does something (a signal) before it can continue
- Use
  - Properly initialize lock and conditional variable

    Example

    ```
    pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;
    pthread_cond_t  cond = PTHREAD_COND_INITIALIZER;
    ```

  - Put calling thread to sleep using `cond_wait()`

    Example

    ```
    Pthread_mutex_lock(&lock);
    while (ready == 0)
        Pthread_cond_wait(&cond, &lock);
    Pthread_mutex_unlock(&lock);
    ```

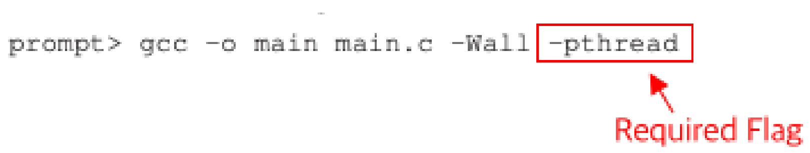
  - Wait until some other thread signals it using `cond_signal()`

    Example

    ```
    Pthread_mutex_lock(&lock);
    ready = 1;
    Pthread_cond_signal(&cond);
    Pthread_mutex_unlock(&lock);
    ```

1.5 Compiling and Running

- Requires a `-pthread` tag



```
prompt> gcc -o main main.c -Wall -pthread
```