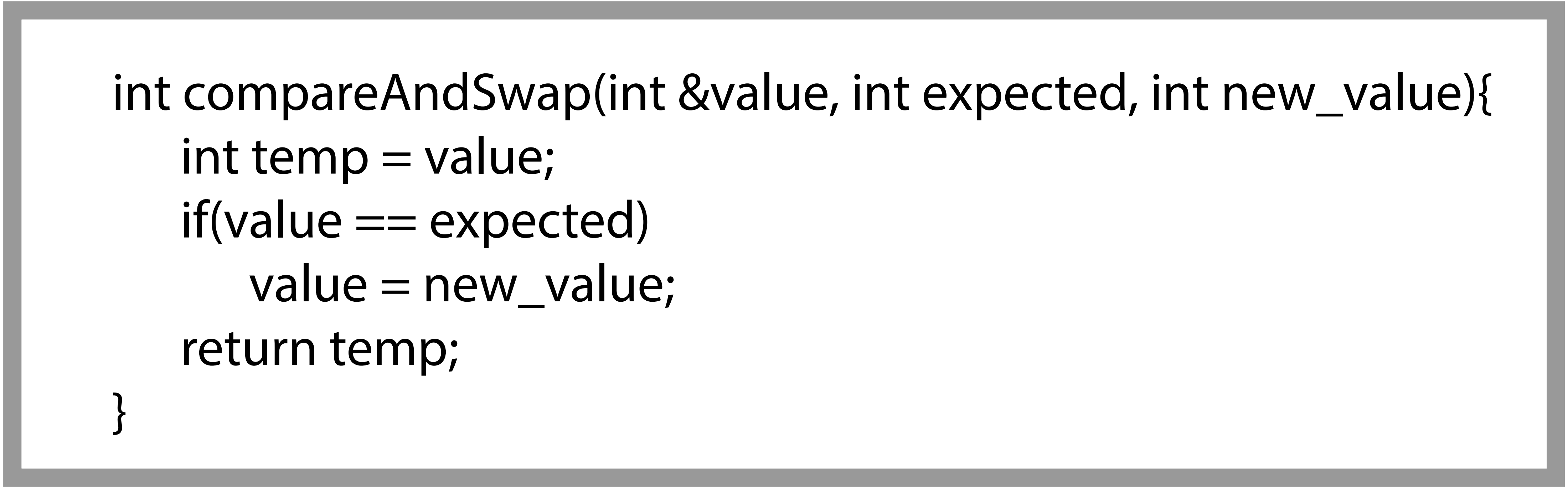
```
bool testAndSet(bool &lock){
        bool value = lock;
                lock = true;
                return value;
}
```

Test and Set



```
int compareAndSwap(int &value, int expected, int new_value){
    int temp = value;
    if(value == expected)
        value = new_value;
    return temp;
}
```

Compare and Swap

Shared Data: bool flag[2] = false;

Process 0

```
while(flag[1]) ;
flag[0] = true;
//Critical section
flag[0] = false;
//Remainder section
```

Process 1

```
while(flag[0]) ;
flag[1] = true;
//Critical section
flag[1] = false;
//Remainder section
```

Lock Variable

Shared Data: turn = 0;

Process 0

```
do{
    while(turn != 0) ;
    //Critical section
    turn = 1
    //Remainder section
}while(1)
```

Process 1

```
do{
    while(turn != 1) ;
    //Critical section
    turn = 0
    //Remainder section
}while(1)
```

Taking Turns

Shared Data: bool flag[2] = false;

Process 0

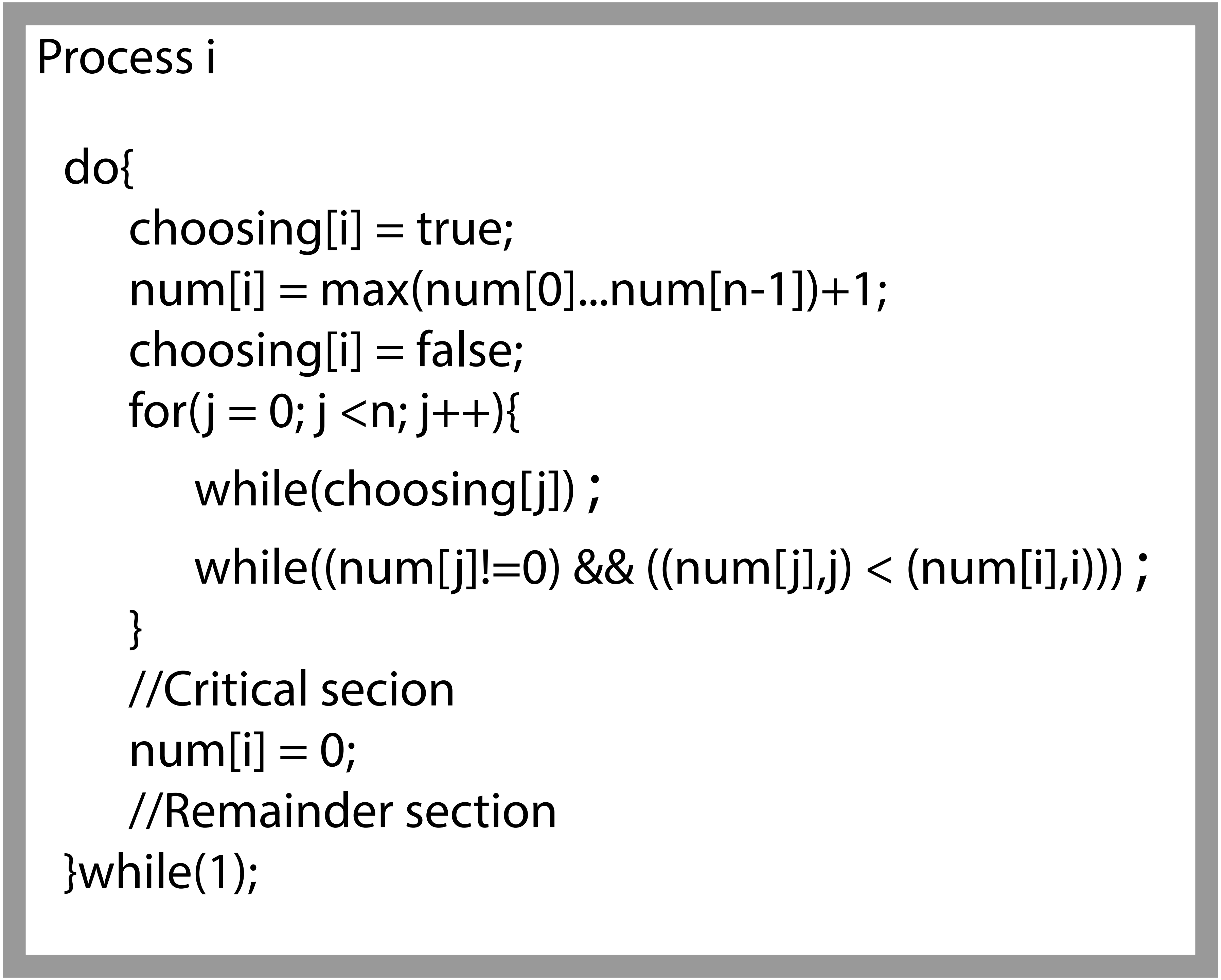
```
do{
    flag[0] = true;
    turn = 1;
    while(flag[1] && turn == 1) ;
    //Critical section
    flag[0] = false;
    //Remainder section
}while(1);
```

Process 1

```
do{
    flag[1] = true;
    turn = 0;
    while(flag[0] && turn == 0) ;
    //Critical section
    flag[1] = false;
    //Remainder section
}while(1);
```

Peterson’s Algorithm

Shared Data: choosing[n]; num[n];



```
Process i

  do{
      choosing[i] = true;
      num[i] = max(num[0]...num[n-1])+1;
      choosing[i] = false;
      for(j = 0; j <n; j++){
          while(choosing[j]) ;
          while((num[j]!=0) && ((num[j],j) < (num[i],i))) ;
      }
      //Critical secion
      num[i] = 0;
      //Remainder section
  }while(1);
```

Bakery Algorithm

Shared Data: available = true;

```
entry_section(){
    while(!available) ;
    available = false;
}
```

```
exit_section(){
    available = true;
}
```

```
do{
    entry_section();
    //Critical section
    exit_section();
    //Remainder section
}while(1);
```

Mutex

Shared Data: semaph = n;

```
wait(semaph){
    while(semaph <= 0) ;
    semaph--;
}
```

```
signal(semaph){
    semaph++;
}
```

Semaphore