Threads with Data

Threads

Data

- Custom data storage with **struct** or **class**
- Wrap a thread with some data
- Program can associate the data with specific threads, less worry about race conditions

Diver struct

thread is created in start

member variable
of an instance of Diver

global variable
still needs a mutex

```
struct Diver{
  int payment = 0;
  thread* exe = NULL;

  void start( ){
    exe = new thread(&Diver::fetch,this);
  }

  void fetch( ){
    while( treasure > 0){
      mtx.lock();
      int amt = min(treasure,13);
      treasure -= amt;
      mtx.unlock()
      payment += amt;
    }
  }

};
```

Diver struct

pointer to *this* instance of Diver

```
struct Diver{
  int payment = 0;
  thread* exe = NULL;

  void start( ){
    exe = new thread( &Diver::fetch , this );
  }

  void fetch( ){
    while(  treasure   > 0){
      mtx.lock();
      int amt = min(treasure,13);
      treasure -= amt;
      mtx.unlock();
      payment += amt;
    }
  }

};
```

starts the thread running

function pointer to the fetch function in Diver

the code

```
#include <iostream>
#include <thread>
#include <vector>
#include <mutex>
using namespace std;
int treasure = 1000;
mutex mtx;

struct Diver{
    int payment = 0;
    thread* exe = NULL;

    void start( ){
        exe = new thread( &Diver::fetch, this);
    }

    void fetch( ){
        while(  treasure   > 0){
            mtx.lock();
            int amt = min(treasure,13);
            treasure -= amt;
            mtx.unlock();
            payment += amt;
        }
    }

};

int main(){
    vector<Diver*> divers;
    for(int i = 0; i < 4; ++i){
        divers.push_back(new Diver);
    }
    for(Diver* d: divers){
        d->start();
    }
    int due = 0;
    for(Diver* d: divers){
        due += d->payment;
    }

    cout << due << endl;
    return 0;
}
```

the code

```
int main(){
  vector<Diver*> divers;
  for(int i = 0; i < 4; ++i){
    divers.push_back(new Diver);
  }

  for(Diver* d: divers){
    d->start();
  }

  int due = 0;
  for(Diver* d: divers){
    due += d->payment;
  }

  cout << due << endl;
  return 0;
}
```

Still missing something?

Each diver has its own payment

the code

```
int main(){
   vector<Diver*> divers;
   for(int i = 0; i < 4; ++i){
      divers.push_back(new Diver);
   }

   for(Diver* d: divers){
      d->start();
   }
```

Need to make sure divers are done!

```
   for(Diver* d: divers){
      d->end();
   }

   int due = 0;
   for(Diver* d: divers){
      due += d->payment;
   }

   cout << due << endl;
   return 0;
}
```

Diver struct

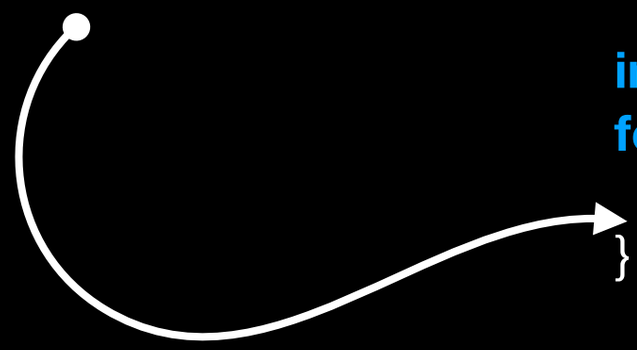
```
struct Diver{
    int payment = 0;
    thread* exe = NULL;

    void start( ){
        exe = new thread( &Diver::fetch  ,  this  );
    }

    void end( ){
        if( exe  != NULL){
            exe->join();
        }
    }

    void fetch( ){
        while(  treasure   > 0){
            mtx.lock();
            int amt = min(treasure,13);
            treasure -= amt;
            mtx.unlock();
            payment += amt;
        }
    }
};
```

use join

the code

```
int main(){
   vector<Diver*> divers;
   for(int i = 0; i < 4; ++i){
      divers.push_back(new Diver);
   }

   for(Diver* d: divers){
      d->start();
   }

   for(Diver* d: divers){
      d->end();
   }

   int due = 0;
   for(Diver* d: divers){
      due += d->payment;
   }

   cout << due << endl;
   return 0;
}
```

Need to delete memory



the code

```
int main(){
   vector<Diver*> divers;
   for(int i = 0; i < 4; ++i){
      divers.push_back(new Diver);
   }

   for(Diver* d: divers){
      d->start();
   }

   for(Diver* d: divers){
      d->end();
   }

   int due = 0;
   for(Diver* d: divers){
      due += d->payment;
      delete d;
   }

   cout << due << endl;
   return 0;
}
```

Diver struct

```
struct Diver{
    int payment = 0;
    thread* exe = NULL;

    ~Diver(){
        delete exe;
    }

    void start( ){
        exe = new thread( &Diver::fetch,this);
    }

    void end( ){
        if( exe  != NULL){
            exe->join();
        }
    }

    void fetch( ){
        while(  treasure  > 0){
            mtx.lock();
            int amt = min(treasure,13);
            treasure -= amt;
            mtx.unlock();
            payment += amt;
        }
    }
};
```

the code

```cpp
#include <iostream>
#include <thread>
#include <vector>
#include <mutex>

using namespace std;
int treasure = 1000;
mutex mtx;

struct Diver{
    int payment = 0;
    thread* exe = NULL;

    ~Diver(){
        delete exe;
    }

    void start( ){
        exe = new thread( &Diver::fetch,this);
    }

    void end( ){
        if( exe  != NULL){
            exe->join();
        }
    }

    void fetch( ){
        while(  treasure  > 0){
            mtx.lock();
            int amt = min(treasure,13);
            treasure -= amt;
            mtx.unlock();
            payment += amt;
        }
    }
};
```

```cpp
int main(){
    vector<Diver*> divers;
    for(int i = 0; i < 4; ++i){
        divers.push_back(new Diver);
    }

    for(Diver* d: divers){
        d->start();
    }

    for(Diver* d: divers){
        d->end();
    }

    int due = 0;
    for(Diver* d: divers){
        due += d->payment;
        delete d;
    }

    cout << due << endl;

    return 0;
}
```