Question 1

1.1 char funnyName[]=”ostrolitch”;

1.2 Because C strings use the function string copy to copy one string to another,

strcpy(FunnyName, FunnyWord);

1.3 If FunnyName and funnyWord are not equal.

1.4 FuunyWord has valid indexes from 0 to 9 only of which index 9 is for the null terminator.

Question 2

2.1 if (num ==0)

2.2 return num + sum (num - 1);

2.3 10

Question 3

3.1 Pointer p2 not initialised (not pointing to any memory address).

3.2 To create a nameless int variable and assigning p3 to pount to it.

3.3 Using the reference operator to assign the address of x to pointer p1

3.4 To specify that p1 and p2 declaration is a pointer

3.5 To produce the variable p point to

3.6 15 15 15

3.7 delete p2;

Question 4

4.1

```
#ifndef CHAMPION_H
#define CHAMPION_H
#include <fstream>
#include <iostream>
#include <cstdlib>
using namespace std;

class Champion
{
   public:
      Champion();
```

```
    Champion( string n,int s,int l);
    ~Champion();
    int get_score();
    friend bool operator >=( const Champion& champ1, const Champion& champ2);
    friend istream& operator >>(istream& ins, const Champion& champ);
    friend ostream & operator << (ostream& outs, const Champion& champ);
private:
        string name;
        int score;
        int level;
};
```

4.2

```
#include "Champion.h"
#include <iostream>
#include <fstream>
#include <cstdlib>
using namespace std;

Champion::Champion()
{
  name = "";
  score = 0;
  level = 0;
}

Champion::Champion( string n,int s,int l)
{
  name = n;
  score = s;
  level = l;
}
```

```
Champion::~Champion(){}

int Champion::get_score(){
return score;
}

bool operator >=( const Champion & champ1,const Champion& champ2){
if (champ1.score >= champ2.score && champ1.level >= champ2.level)
   return true;
else
   return false;
}

istream& operator >>(istream& ins,  Champion& champ){
ins >> champ.name;
ins >> champ.score;
ins >> champ.level;
return ins;
}

ostream & operator << (ostream& outs, const Champion& champ){
outs << champ.name << "\t"<<champ.score << "\t" <<champ.level << endl;
return outs;
}
```

4.3

```
#include "Champion.h"
#include <iostream>
#include <fstream>
#include <cstdlib>
using namespace std;
```

```
int main()
{
   ifstream fin;
   int champScore, champLevel,score,level;
   string name;

   cout << "Enter champion's score "<< endl;
   cin >> champScore;

   cout << "Enter champion's level "<< endl;
   cin >> champLevel;

   Champion current_champion("",champScore, champLevel);

   fin.open("winners.txt");
   if(fin.fail()){
      cout <<"Input file failed to open " << endl;
      exit(1);
   }

   while(!fin.eof()){
         fin >> name >>score >> level;
      Champion winner(name,score, level);

   if(winner >= current_champion){

      cout << winner;
   }

}
```

```
fin.close();
return 0;
}
```

Question 5

5.1

```
#ifndef TRAVEL_H
#define TRAVEL_H



class Travel
{
   public:
      Travel();
      Travel(string name, string passport, string toFlightNr, string returnFlghtNr, string toDate, string
returnDate);
      void set_details(string name, string Passport);
      void set_flight(string toFlightNr, string returnFlightNr, string toDate, string returnDate);
      void display_info() const;



   protected:


   private:
      string Name;
      string Passport;
      string TODate;
      string RETDate;
      string TONumber;
      string RETNumber;
};
```

```
#endif // TRAVEL_H
```

5.2

```
#ifndef SATOAFRICA_H
#define SATOAFRICA_H


#include "Travel.h"


class SAtoAFRICA : public Travel
{
   public:
      SAtoAFRICA();
      SAtoAFRICA(string nameP, string passportNr, string toFlightNr, string returnFlghtNr, string
toDate, string returnDate, string YFP): Travel(nameP,passportNr,toFlightNr,returnFlghtNr, toDate,
returnDate)
      void display_info() const;


   private:
      string YF;
};


#endif // SATOAFRICA_H
```

5.3

An **overloaded** function is a function that shares its name with one or more other functions, but which has a different parameter list. The compiler chooses which function is desired based upon the arguments used.
An **overridden** function is a method in a descendant class that has a different definition than a virtual function in an ancestor class. The compiler chooses which function is desired based upon the type of the object being used to call the function.
A **redefined** function is a method in a descendant class that has a different definition than a non-virtual function in an ancestor class. Don't do this. Since the method is not virtual, the compiler chooses which function to call based upon the static type of the object reference rather than the actual type of the object.

Question 6

6.1

```
#include <iostream>
```

```
using namespace std;

template <class TCode ,class TNumber>
class Directory{
public :
   Directory();
   void AddPart(TCode code, const TNumber &number);
   TNumber Check(TCode code) const;
private:
   vector<TCode> Code;
   vector<TNumber> Number;
};
```

6.2

```
template <class TCode ,class TNumber>
TNumber Directory<TCode,TNumber>::Check(TCode code) const
{
   assert(code > 0);
   for(int i = 0; i < code.size(); i++){
    if( code == code[i])
      return Number[i];
    else
      return 0;
}
```

6.3

```
Directory<string,double> d1();
```