Output Formatting

What If We Wanted to Print Within a Class?

```
class MyCout
{
  MyCout();
  void SetFixed();
  void SetPrecision(int value);
  ...
  void Print(double d);
}

int main()
{
  double d = 5.0;
  cout << d << endl;
  MyCout c;
  c.SetFixed();
  c.SetPrecision(5);
  c.Print(d);
  cout << d << endl;;
}
```

- What would happen once we want to print outside of the class?

What If We Wanted to Print Within a Class?

```
class MyCout
{
  MyCout();
  void SetFixed();
  void SetPrecision(int value);
  ...
  void Print(double d);
}

int main()
{
  double d = 5.0;
  cout << d << endl;
  MyCout c;
  c.SetFixed();
  c.SetPrecision(5);
  c.Print(d)
  cout << d << endl;
}
```

```
Output:
5
5.00000
5.00000
```

- The obvious thing would be to update cout, but this affects everyone else who uses the object
- This means we need to save and restore the state of cout

Preserving cout

```
EXEC = driver.x

build:
    g++ -o $(EXEC) driver.cpp my_cout.cpp

run: build
    ./$(EXEC)

clean:
    rm -f $(EXEC)
```

Variable declaration

Preserving cout

```
EXEC = driver.x

build:
    g++ -o $(EXEC) driver.cpp my_cout.cpp

run: build
    ./$(EXEC)

clean:
    rm -f $(EXEC)
```

Variable use

Preserving cout

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

        void SetDefault();
        void SetFixed();
        void SetPrecision(int value);
        void SetWidth(int value);
        void SetFill(char c);

        void Print(double d);

    private:
        int format;
        int precision;
        int width;
        char fill;

        static const int FORMAT_DEFAULT = 0;
        static const int FORMAT_FIXED = 1;
};
```

Change State

Special Print

Store Changed State

Preserving cout

```
MyCout::MyCout() {
    format = FORMAT_DEFAULT;
    precision = 0;
    width = 0;
    fill = ' ';
}
void MyCout::SetDefault() {
    format = FORMAT_DEFAULT;
}
void MyCout::SetFixed() {
    format = FORMAT_FIXED;
}
void MyCout::SetPrecision(int value) {
    precision = value;
}
void MyCout::SetWidth(int value) {
    width = value;
}
void MyCout::SetFill(char c) {
    fill = c;
}
```

Change state mutators just store the change

Preserving cout

```
MyCout::MyCout() {
    format = FORMAT_DEFAULT;
    precision = 0;
    width = 0;
    fill = ' ';
}
void MyCout::SetDefault() {
    format = FORMAT_DEFAULT;
}
void MyCout::SetFixed() {
    format = FORMAT_FIXED;
}
void MyCout::SetPrecision(int value) {
    precision = value;
}
void MyCout::SetWidth(int value) {
    width = value;
}
void MyCout::SetFill(char c) {
    fill = c;
}
```

Sets the max number of digits to display...

Preserving cout

```
MyCout::MyCout() {
    format = FORMAT_DEFAULT;
    precision = 0;
    width = 0;
    fill = ' ';
}
void MyCout::SetDefault() {
    format = FORMAT_DEFAULT;
}
void MyCout::SetFixed() {
    format = FORMAT_FIXED;
}
void MyCout::SetPrecision(int value) {
    precision = value;
}
void MyCout::SetWidth(int value) {
    width = value;
}
void MyCout::SetFill(char c) {
    fill = c;
}
```

Sets the max number of digits to display...

Max is total number

Preserving cout

```
MyCout::MyCout() {
    format = FORMAT_DEFAULT;
    precision = 0;
    width = 0;
    fill = ' ';
}
void MyCout::SetDefault() {
    format = FORMAT_DEFAULT;
}
void MyCout::SetFixed() {
    format = FORMAT_FIXED;
}
void MyCout::SetPrecision(int value) {
    precision = value;
}
void MyCout::SetWidth(int value) {
    width = value;
}
void MyCout::SetFill(char c) {
    fill = c;
}
```

Sets the max number of digits to display...

Max is after decimal point

Preserving cout

```
MyCout::MyCout() {
    format = FORMAT_DEFAULT;
    precision = 0;
    width = 0;
    fill = ' ';
}
void MyCout::SetDefault() {
    format = FORMAT_DEFAULT;
}
void MyCout::SetFixed() {
    format = FORMAT_FIXED;
}
void MyCout::SetPrecision(int value) {
    precision = value;
}
void MyCout::SetWidth(int value) {
    width = value;
}
void MyCout::SetFill(char c) {
    fill = c;
}
```

Pads the display with the width number of fill characters

Preserving cout

```
MyCout::MyCout() {
    format = FORMAT_DEFAULT;
    precision = 0;
    width = 0;
    fill = ' ';
}
void MyCout::SetDefault() {
    format = FORMAT_DEFAULT;
}
void MyCout::SetFixed() {
    format = FORMAT_FIXED;
}
void MyCout::SetPrecision(int value) {
    precision = value;
}
void MyCout::SetWidth(int value) {
    width = value;
}
void MyCout::SetFill(char c) {
    fill = c;
}
```

Fill character
for field width

Preserving cout

```
void MyCout::Print(double d) {
    //save old flags
    ios_base::fmtflags fl = cout.flags();
    int w = cout.width();
    int f = cout.fill();
    int p = cout.precision();

    //set our flags
    cout.unsetf(ios_base::floatfield); //removes all formatting
    if (format == FORMAT_FIXED)
        cout.setf(ios_base::fixed);
    cout.width(width);
    cout.fill(fill);
    cout.precision(precision);

    //print
    cout << d << endl;

    //reset old flags
    cout.flags(fl);
    cout.width(w);
    cout.fill(f);
    cout.precision(p);
}
```

Print does all the work

Preserving cout

```
void MyCout::Print(double d) {
    //save old flags
    ios_base::fmtflags fl = cout.flags();
    int w = cout.width();
    int f = cout.fill();
    int p = cout.precision();

    //set our flags
    cout.unsetf(ios_base::floatfield); //removes all formatting
    if (format == FORMAT_FIXED)
        cout.setf(ios_base::fixed);
    cout.width(width);
    cout.fill(fill);
    cout.precision(precision);

    //print
    cout << d << endl;

    //reset old flags
    cout.flags(fl);
    cout.width(w);
    cout.fill(f);
    cout.precision(p);
}
```

Save old state
into local variables

Preserving cout

```
void MyCout::Print(double d) {
    //save old flags
    ios_base::fmtflags fl = cout.flags();
    int w = cout.width();
    int f = cout.fill();
    int p = cout.precision();

    //set our flags
    cout.unsetf(ios_base::floatfield); //removes all formatting
    if (format == FORMAT_FIXED)
        cout.setf(ios_base::fixed);
    cout.width(width);
    cout.fill(fill);
    cout.precision(precision);

    //print
    cout << d << endl;

    //reset old flags
    cout.flags(fl);
    cout.width(w);
    cout.fill(f);
    cout.precision(p);
}
```

Replace with our state

Preserving cout

```
void MyCout::Print(double d) {
    //save old flags
    ios_base::fmtflags fl = cout.flags();
    int w = cout.width();
    int f = cout.fill();
    int p = cout.precision();

    //set our flags
    cout.unsetf(ios_base::floatfield); //removes all formatting
    if (format == FORMAT_FIXED)
        cout.setf(ios_base::fixed);
    cout.width(width);
    cout.fill(fill);
    cout.precision(precision);

    //print
    cout << d << endl;

    //reset old flags
    cout.flags(fl);
    cout.width(w);
    cout.fill(f);
    cout.precision(p);
}
```

Actual printing

(or abstractly, whatever the function needs to do)

Preserving cout

```
void MyCout::Print(double d) {
    //save old flags
    ios_base::fmtflags fl = cout.flags();
    int w = cout.width();
    int f = cout.fill();
    int p = cout.precision();

    //set our flags
    cout.unsetf(ios_base::floatfield); //removes all formatting
    if (format == FORMAT_FIXED)
        cout.setf(ios_base::fixed);
    cout.width(width);
    cout.fill(fill);
    cout.precision(precision);

    //print
    cout << d << endl;

    //reset old flags
    cout.flags(fl);
    cout.width(w);
    cout.fill(f);
    cout.precision(p);
}
```

Restore the saved state

Preserving cout

```
void MyCout::Print(double d) {
    //save old flags
    ios_base::fmtflags fl = cout.flags();
    int w = cout.width();
    int f = cout.fill();
    int p = cout.precision();

    //set our flags
    cout.unsetf(ios_base::floatfield); //removes all formatting
    if (format == FORMAT_FIXED)
        cout.setf(ios_base::fixed);
    cout.width(width);
    cout.fill(fill);
    cout.precision(precision);

    //print
    cout << d << endl;

    //reset old flags
    cout.flags(fl);
    cout.width(w);
    cout.fill(f);
    cout.precision(p);
}
```

This paradigm comes up whenever a container is shared between multiple scopes

e.g. function calls at the assembler level need to save/restore register values

Preserving cout

```
void special_print(double num) {
    MyCout mc;
    mc.SetFixed();
    mc.SetWidth(12);
    mc.SetPrecision(3);
    mc.SetFill('-');
    mc.Print(num);
}
void example(double num) {
    cout << num << endl;
    special_print(num);
    cout << num << endl;
    cout << endl;
}
int main() {
    //set main cout flags
    cout << scientific;

    //numbers to print
    example(10);
    example(56.45);
    example(1.0 / 3.0);
    example(INFINITY);
}
```

Handles setting various flags

Shows that state has been restored

Tests multiple different scenarios

Preserving cout

```
void special_print(double num) {
    MyCout mc;
    mc.SetFixed();
    mc.SetWidth(12);
    mc.SetPrecision(3);
    mc.SetFill('-');
    mc.Print(num);
}
void example(double num) {
    cout << num << endl;
    special_print(num);
    cout << num << endl;
    cout << endl;
}
int main() {
    //set main cout flags
    cout << scientific;

    //numbers to print
    example(10);
    example(56.45);
    example(1.0 / 3.0);
    example(INFINITY);
}
```

12 characters

```
./driver.x
1.000000e+01
------10.000
1.000000e+01

5.645000e+01
------56.450
5.645000e+01

3.333333e-01
-------0.333
3.333333e-01

inf
---------inf
inf
```

Preserving cout

```
void special_print(double num) {
    MyCout mc;
    mc.SetFixed();
    mc.SetWidth(12);
    mc.SetPrecision(3);
    mc.SetFill('-');
    mc.Print(num);
}
void example(double num) {
    cout << num << endl;
    special_print(num);
    cout << num << endl;
    cout << endl;
}
int main() {
    //set main cout flags
    cout << scientific;

    //numbers to print
    example(10);
    example(56.45);
    example(1.0 / 3.0);
    example(INFINITY);
}
```

Padded with -

```
./driver.x
1.000000e+01
------10.000
1.000000e+01

5.645000e+01
------56.450
5.645000e+01

3.333333e-01
-------0.333
3.333333e-01

inf
---------inf
inf
```

Preserving cout

```
void special_print(double num) {
    MyCout mc;
    mc.SetFixed();
    mc.SetWidth(12);
    mc.SetPrecision(3);
    mc.SetFill('-');
    mc.Print(num);
}
void example(double num) {
    cout << num << endl;
    special_print(num);
    cout << num << endl;
    cout << endl;
}
int main() {
    //set main cout flags
    cout << scientific;

    //numbers to print
    example(10);
    example(56.45);
    example(1.0 / 3.0);
    example(INFINITY);
}
```

Adds 3 0s to 10

```
./driver.x
1.000000e+01
------10.000
1.000000e+01

5.645000e+01
------56.450
5.645000e+01

3.333333e-01
-------0.333
3.333333e-01

inf
---------inf
inf
```

Preserving cout

```
void special_print(double num) {
    MyCout mc;
    mc.SetFixed();
    mc.SetWidth(12);
    mc.SetPrecision(3);
    mc.SetFill('-');
    mc.Print(num);
}
void example(double num) {
    cout << num << endl;
    special_print(num);
    cout << num << endl;
    cout << endl;
}
int main() {
    //set main cout flags
    cout << scientific;

    //numbers to print
    example(10);
    example(56.45);
    example(1.0 / 3.0);
    example(INFINITY);
}
```

Adds 1 0s to 56.45

```
./driver.x
1.000000e+01
------10.000
1.000000e+01

5.645000e+01
------56.450
5.645000e+01

3.333333e-01
-------0.333
3.333333e-01

inf
---------inf
inf
```

Preserving cout

```
void special_print(double num) {
    MyCout mc;
    mc.SetFixed();
    mc.SetWidth(12);
    mc.SetPrecision(3);
    mc.SetFill('-');
    mc.Print(num);
}
void example(double num) {
    cout << num << endl;
    special_print(num);
    cout << num << endl;
    cout << endl;
}
int main() {
    //set main cout flags
    cout << scientific;

    //numbers to print
    example(10);
    example(56.45);
    example(1.0 / 3.0);
    example(INFINITY);
}
```

Truncates 0.3333...

```
./driver.x
1.000000e+01
------10.000
1.000000e+01

5.645000e+01
------56.450
5.645000e+01

3.333333e-01
-------0.333
3.333333e-01

inf
---------inf
inf
```

Preserving cout

```
void special_print(double num) {
    MyCout mc;
    mc.SetFixed();
    mc.SetWidth(12);
    mc.SetPrecision(3);
    mc.SetFill('-');
    mc.Print(num);
}
void example(double num) {
    cout << num << endl;
    special_print(num);
    cout << num << endl;
    cout << endl;
}
int main() {
    //set main cout flags
    cout << scientific;

    //numbers to print
    example(10);
    example(56.45);
    example(1.0 / 3.0);
    example(INFINITY);
}
```

Float has special cases that can't change print format

Other big one is NaN

```
./driver.x
1.000000e+01
------10.000
1.000000e+01

5.645000e+01
------56.450
5.645000e+01

3.333333e-01
-------0.333
3.333333e-01

inf
---------inf
inf
```

Backups

- Several people had some non-implementation problems with assignment 1
  - Getting files off the server to submit
  - Accidentally deleting the wrong file
- Here, I'll go over ways you can use Makefiles to help with this
  - Local archival
  - Email backups
- There are many other techniques you can use; here are a few more. See if you can get them to work in your solution
  - Safe remove (instead of rm, mv to a ~/.trash folder)
  - Remote copy (scp, secure like sftp but automated like wget)
  - Remote transfer (automated stfp using expect)

Backups

```
EXEC = driver.x
BACKUP_FILES = driver.cpp my_cout.cpp my_cout.h Makefile

EMAIL_RECIPIENTS = dennis@cs.fsu.edu bbd09@my.fsu.edu
EMAIL_SUBJECT = "Backup Test"
EMAIL_MESSAGE = "This is an example illustrating use of email backups in Makefiles."
EMAIL_ATTACHMENT = temp.tar

build:
    g++ -o $(EXEC) driver.cpp my_cout.cpp

run: build
    ./$(EXEC)

archive: clean
    tar cf `date "+backup_%F_%H_%M_%S.tar"` $(BACKUP_FILES)

email: clean
    tar cf $(EMAIL_ATTACHMENT) $(BACKUP_FILES)
    echo $(EMAIL_MESSAGE) | mail -a $(EMAIL_ATTACHMENT) -s $(EMAIL_SUBJECT) $
(EMAIL_RECIPIENTS)
    rm -f $(EMAIL_ATTACHMENT)

clean:
    rm -f $(EXEC)  *~
```

Create a local backup

Useful if you want to checkpoint progress but don't want to clutter other machines or if the implementation is incomplete

Backups

```
EXEC = driver.x
BACKUP_FILES = driver.cpp my_cout.cpp my_cout.h Makefile

EMAIL_RECIPIENTS = dennis@cs.fsu.edu bbd09@my.fsu.edu
EMAIL_SUBJECT = "Backup Test"
EMAIL_MESSAGE = "This is an example illustrating use of email backups in Makefiles."
EMAIL_ATTACHMENT = temp.tar

build:
    g++ -o $(EXEC) driver.cpp my_cout.cpp

run: build
    ./$(EXEC)

archive: clean
    tar cf `date "+backup_%F_%H_%M_%S.tar"` $(BACKUP_FILES)

email: clean
    tar cf $(EMAIL_ATTACHMENT) $(BACKUP_FILES)
    echo $(EMAIL_MESSAGE) | mail -a $(EMAIL_ATTACHMENT) -s $(EMAIL_SUBJECT) $
(EMAIL_RECIPIENTS)
    rm -f $(EMAIL_ATTACHMENT)

clean:
    rm -f $(EXEC)  *~
```

Create a local backup

Here you tar each of the files you want to save

You could also tar directories, however don't tar the directory containing the archives

Backups

```
EXEC = driver.x
BACKUP_FILES = driver.cpp my_cout.cpp my_cout.h Makefile

EMAIL_RECIPIENTS = dennis@cs.fsu.edu bbd09@my.fsu.edu
EMAIL_SUBJECT = "Backup Test"
EMAIL_MESSAGE = "This is an example illustrating use of email backups in Makefiles."
EMAIL_ATTACHMENT = temp.tar

build:
    g++ -o $(EXEC) driver.cpp my_cout.cpp

run: build
    ./$(EXEC)

archive: clean
    tar cf `date "+backup_%F_%H_%M_%S.tar"` $(BACKUP_FILES)

email: clean
    tar cf $(EMAIL_ATTACHMENT) $(BACKUP_FILES)
    echo $(EMAIL_MESSAGE) | mail -a $(EMAIL_ATTACHMENT) -s $(EMAIL_SUBJECT) $
(EMAIL_RECIPIENTS)
    rm -f $(EMAIL_ATTACHMENT)

clean:
    rm -f $(EXEC)  *~
```

Create a local backup

To make sure the file is unique
you can use the date command

This example will cause files to look like:
backup_2016-09-28_12_44_38.tar

Backups

```
EXEC = driver.x
BACKUP_FILES = driver.cpp my_cout.cpp my_cout.h Makefile

EMAIL_RECIPIENTS = dennis@cs.fsu.edu bbd09@my.fsu.edu
EMAIL_SUBJECT = "Backup Test"
EMAIL_MESSAGE = "This is an example illustrating use of email backups in Makefiles."
EMAIL_ATTACHMENT = temp.tar

build:
    g++ -o $(EXEC) driver.cpp my_cout.cpp

run: build
    ./$(EXEC)

archive: clean
    tar cf `date "+backup_%F_%H_%M_%S.tar"` $(BACKUP_FILES)

email: clean
    tar cf $(EMAIL_ATTACHMENT) $(BACKUP_FILES)
    echo $(EMAIL_MESSAGE) | mail -a $(EMAIL_ATTACHMENT) -s $(EMAIL_SUBJECT) $
(EMAIL_RECIPIENTS)
    rm -f $(EMAIL_ATTACHMENT)

clean:
    rm -f $(EXEC)  *~
```

Create a remote backup

This is useful for getting files off the server or versioning your code

Backups

```
EXEC = driver.x
BACKUP_FILES = driver.cpp my_cout.cpp my_cout.h Makefile

EMAIL_RECIPIENTS = dennis@cs.fsu.edu bbd09@my.fsu.edu
EMAIL_SUBJECT = "Backup Test"
EMAIL_MESSAGE = "This is an example illustrating use of email backups in Makefiles."
EMAIL_ATTACHMENT = temp.tar

build:
    g++ -o $(EXEC) driver.cpp my_cout.cpp

run: build
    ./$(EXEC)

archive: clean
    tar cf `date "+backup_%F_%H_%M_%S.tar"` $(BACKUP_FILES)

email: clean
    tar cf $(EMAIL_ATTACHMENT) $(BACKUP_FILES)
    echo $(EMAIL_MESSAGE) | mail -a $(EMAIL_ATTACHMENT) -s $(EMAIL_SUBJECT) $
(EMAIL_RECIPIENTS)
    rm -f $(EMAIL_ATTACHMENT)

clean:
    rm -f $(EXEC)  *~
```

Create a remote backup

This specific example assumes you are on a cs server (e.g. shell)

This only works if the local environment Has mail settings configured

Backups

```
EXEC = driver.x
BACKUP_FILES = driver.cpp my_cout.cpp my_cout.h Makefile

EMAIL_RECIPIENTS = dennis@cs.fsu.edu bbd09@my.fsu.edu
EMAIL_SUBJECT = "Backup Test"
EMAIL_MESSAGE = "This is an example illustrating use of email backups in Makefiles."
EMAIL_ATTACHMENT = temp.tar

build:
    g++ -o $(EXEC) driver.cpp my_cout.cpp

run: build
    ./$(EXEC)

archive: clean
    tar cf `date "+backup_%F_%H_%M_%S.tar"` $(BACKUP_FILES)

email: clean
    tar cf $(EMAIL_ATTACHMENT) $(BACKUP_FILES)
    echo $(EMAIL_MESSAGE) | mail -a $(EMAIL_ATTACHMENT) -s $(EMAIL_SUBJECT) $
(EMAIL_RECIPIENTS)
    rm -f $(EMAIL_ATTACHMENT)

clean:
    rm -f $(EXEC)  *~
```

Create a remote backup

First tar's the files into a temp file

And removes the temp file when done

Backups

```
EXEC = driver.x
BACKUP_FILES = driver.cpp my_cout.cpp my_cout.h Makefile

EMAIL_RECIPIENTS = dennis@cs.fsu.edu bbd09@my.fsu.edu
EMAIL_SUBJECT = "Backup Test"
EMAIL_MESSAGE = "This is an example illustrating use of email backups in Makefiles."
EMAIL_ATTACHMENT = temp.tar

build:
    g++ -o $(EXEC) driver.cpp my_cout.cpp

run: build
    ./$(EXEC)

archive: clean
    tar cf `date "+backup_%F_%H_%M_%S.tar"` $(BACKUP_FILES)

email: clean
    tar cf $(EMAIL_ATTACHMENT) $(BACKUP_FILES)
    echo $(EMAIL_MESSAGE) | mail -a $(EMAIL_ATTACHMENT) -s $(EMAIL_SUBJECT) $
(EMAIL_RECIPIENTS)
    rm -f $(EMAIL_ATTACHMENT)

clean:
    rm -f $(EXEC)  *~
```

Create a remote backup

To send the email, use the mail command

Backups

```
EXEC = driver.x
BACKUP_FILES = driver.cpp my_cout.cpp my_cout.h Makefile

EMAIL_RECIPIENTS = dennis@cs.fsu.edu bbd09@my.fsu.edu
EMAIL_SUBJECT = "Backup Test"
EMAIL_MESSAGE = "This is an example illustrating use of email backups in Makefiles."
EMAIL_ATTACHMENT = temp.tar

build:
    g++ -o $(EXEC) driver.cpp my_cout.cpp

run: build
    ./$(EXEC)

archive: clean
    tar cf `date "+backup_%F_%H_%M_%S.tar"` $(BACKUP_FILES)

email: clean
    tar cf $(EMAIL_ATTACHMENT) $(BACKUP_FILES)
    echo $(EMAIL_MESSAGE) | mail -a $(EMAIL_ATTACHMENT) -s $(EMAIL_SUBJECT) $
(EMAIL_RECIPIENTS)
    rm -f $(EMAIL_ATTACHMENT)

clean:
    rm -f $(EXEC)  *~
```

Create a remote backup

To send the email, use the mail command
-a for an attachment

Backups

```
EXEC = driver.x
BACKUP_FILES = driver.cpp my_cout.cpp my_cout.h Makefile

EMAIL_RECIPIENTS = dennis@cs.fsu.edu bbd09@my.fsu.edu
EMAIL_SUBJECT = "Backup Test"
EMAIL_MESSAGE = "This is an example illustrating use of email backups in Makefiles."
EMAIL_ATTACHMENT = temp.tar

build:
    g++ -o $(EXEC) driver.cpp my_cout.cpp

run: build
    ./$(EXEC)

archive: clean
    tar cf `date "+backup_%F_%H_%M_%S.tar"` $(BACKUP_FILES)

email: clean
    tar cf $(EMAIL_ATTACHMENT) $(BACKUP_FILES)
    echo $(EMAIL_MESSAGE) | mail -a $(EMAIL_ATTACHMENT) -s $(EMAIL_SUBJECT) $
(EMAIL_RECIPIENTS)
    rm -f $(EMAIL_ATTACHMENT)

clean:
    rm -f $(EXEC)  *~
```

Create a remote backup

To send the email, use the mail command
-s for the email subject

Backups

```
EXEC = driver.x
BACKUP_FILES = driver.cpp my_cout.cpp my_cout.h Makefile

EMAIL_RECIPIENTS = dennis@cs.fsu.edu bbd09@my.fsu.edu
EMAIL_SUBJECT = "Backup Test"
EMAIL_MESSAGE = "This is an example illustrating use of email backups in Makefiles."
EMAIL_ATTACHMENT = temp.tar

build:
    g++ -o $(EXEC) driver.cpp my_cout.cpp

run: build
    ./$(EXEC)

archive: clean
    tar cf `date "+backup_%F_%H_%M_%S.tar"` $(BACKUP_FILES)

email: clean
    tar cf $(EMAIL_ATTACHMENT) $(BACKUP_FILES)
    echo $(EMAIL_MESSAGE) | mail -a $(EMAIL_ATTACHMENT) -s $(EMAIL_SUBJECT) $
(EMAIL_RECIPIENTS)
    rm -f $(EMAIL_ATTACHMENT)

clean:
    rm -f $(EXEC)  *~
```

Create a remote backup

To send the email, use the mail command
Mail body comes from stdin
(can use echo or a file)

Backups

```
EXEC = driver.x
BACKUP_FILES = driver.cpp my_cout.cpp my_cout.h Makefile

EMAIL_RECIPIENTS = dennis@cs.fsu.edu bbd09@my.fsu.edu
EMAIL_SUBJECT = "Backup Test"
EMAIL_MESSAGE = "This is an example illustrating use of email backups in Makefiles."
EMAIL_ATTACHMENT = temp.tar

build:
    g++ -o $(EXEC) driver.cpp my_cout.cpp

run: build
    ./$(EXEC)

archive: clean
    tar cf `date "+backup_%F_%H_%M_%S.tar"` $(BACKUP_FILES)

email: clean
    tar cf $(EMAIL_ATTACHMENT) $(BACKUP_FILES)
    echo $(EMAIL_MESSAGE) | mail -a $(EMAIL_ATTACHMENT) -s $(EMAIL_SUBJECT) $
(EMAIL_RECIPIENTS)
    rm -f $(EMAIL_ATTACHMENT)

clean:
    rm -f $(EXEC)  *~
```

Create a remote backup

To send the email, use the mail command
After the arguments come the mail recipients