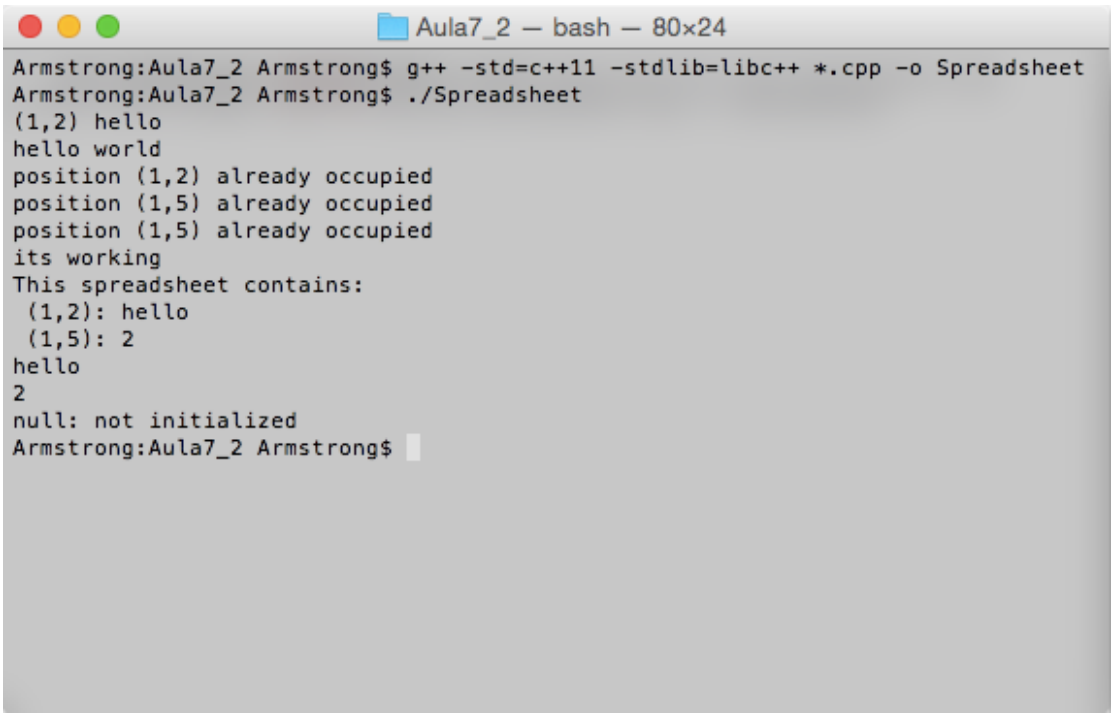
Aula 7 – Exercício de C++
Spreadsheet



AddStrings.cpp

```
#include "AddStrings.h"

AddStrings::AddStrings(int line, int column, String first, String second):
DynamicCell<std::string>(line, column, first, second) {

    std::string s = first.getValue() + second.getValue();
    setValue(s);
}
```

AddStrings.h

```
#ifndef __ADDSTRINGS_H__
#define __ADDSTRINGS_H__

#include "DynamicCell.h"

// this is a formula that adds strings: example that formulas are working.
class AddStrings: public DynamicCell<std::string>{

public:
    AddStrings(int line, int column, String first, String second);
};

#endif
```

Cell.h

```
#ifndef __CELL_H__
#define __CELL_H__


class Cell{

private:
    int _id = 0;

public:

    virtual ~Cell() {}

    void setAsString();

    void setAsInteger();

    bool isString();

    bool isInteger();

};

#endif
```

Content.h

```
#ifndef __CONTENT_H__
#define __CONTENT_H__

#include "Coordinates.h"
#include <string>
#include <iostream>
#include <utility>
#include "Cell.h"

// because Integers and Strings work in the same way except in their type
template <class T>
class Content: public Cell{

    T _value;
    Coordinates _coor;

public:

    Content(int line, int column, T value): _coor(line, column), _value(value) {}

    void setValue(T value) { _value = value; };

    T getValue() { return _value; };

    Coordinates getCoordinates() { return _coor; };

    void setCoordinates(int line, int column) { _coor(line, column); }


};

inline bool operator< (Content<std::string> s1, Content<std::string> s2){ return 0 <
0; }

inline bool operator< (Content<int> s1, Content<int> s2){ return s1.getValue() <
s2.getValue(); }

inline bool operator< (Content<int> s1, Content<std::string> s2){ return s1.getValue()
< 0; }

inline bool operator< (Content<std::string> s1, Content<int> s2){ return 0 <
s2.getValue(); }


#endif
```

Coordinates.cpp

```
#include "Coordinates.h"


Coordinates::Coordinates(int line, int column) {_coordenadas = std::make_pair(line,
column);}

int Coordinates::getLine() { return _coordenadas.first; }

int Coordinates::getColumn() { return _coordenadas.second;}

std::string Coordinates::toString() {
    return "(" + std::to_string(getLine()) + "," + std::to_string(getColumn()) + ")";
}
```

Coordinates.h

```
#ifndef __COORDINATES_H__
#define __COORDINATES_H__

#include <iostream>
#include <string>

class Coordinates
{
private:
    std::pair<int, int> _coordenadas;

public:
    Coordinates(int line, int column);

    int getLine();

    int getColumn();

    std::string toString();

};

#endif
```

Integer.cpp

```
#include "Integer.h"

Integer::Integer(int line, int column, int i): Content(line,column,i) {
setAsInteger(); }

std::ostream& operator<<(std::ostream& os, Integer& dt)
{
    os << dt.getCoordinates().toString() << " " << dt.getValue() << std::endl;
    return os;
}
```

Integer.h

```
#ifndef __INTEGER_H__
#define __INTEGER_H__

#include "Content.h"


class Integer: public Content<int>{

public:

    Integer(int line, int column, int i);

    friend std::ostream& operator<<(std::ostream& os, Integer& dt);

};

#endif
```

DynamicCell.h

```
#ifndef __DynamicCell_H__
#define __DynamicCell_H__

#include "Content.h"
#include "String.h"
#include "Integer.h"

template <class T>
class DynamicCell:  public Content<T>, public Cell{ // used for references and
formulas

private:
    std::string _firstString = "";
    std::string _secondString = "";
    int _firstInt = 0;
    int _secondInt = 0;

public:

    // for formulas with 2 Strings
    DynamicCell<std::string>(int line, int column, String s1, String s2):
    Content<std::string>(line, column, ""), _firstString(s1.getValue()),
_secondString(s2.getValue()) {}

    // for formulas with 2 Integers
    DynamicCell<int>(int line, int column, Integer i1, Integer i2):
    Content<int>(line, column, 0), _firstInt(i1.getValue()), _secondInt(i2.getValue())
{}


    // REFERENCES
    DynamicCell<std::string>(int line, int column, String s):
    Content<std::string>(line, column, ""), _firstString(s.getValue()) {}

    DynamicCell<int>(int line, int column, Integer i):
    Content<int>(line, column, 0), _firstInt(i.getValue()) {}

    // getters
    template <typename C>
    C getFirst() {
        if (std::is_same<C, int>::value)
            return _firstInt;
        return _firstString;
    }

    template <typename C>
    C getSecond() {

        if (std::is_same<C, int>::value)
            return _secondInt;
        return _secondString;
    }


};

#endif
```

Spreadsheet.h

```
#include <unordered_map>
#include "String.h"
#include "Integer.h"
#include "DynamicCell.h"
#include "AddStrings.h"
#include <utility>
#include <iostream>

class Spreadsheet{

private:
    // used unordered_maps because a matrix of vectors or arrays would
    // take up too much space

    std::unordered_map<std::string, String&> _spreadsheetString;
    std::unordered_map<std::string, Integer&> _spreadsheetInteger;
    std::unordered_map<std::string, Cell&> _spreadsheetDCell;


public:


    template <typename T>
    void add(T);

    std::string getContent(int line, int column);

    bool freePosition(Coordinates _coord);

    void print();

    friend std::ostream& operator<<(std::ostream& os, Spreadsheet obj);

    std::string toString();

    bool freeString(Coordinates _coord);

    bool freeInteger(Coordinates _coord);

    bool freeDCell(Coordinates _coord);
};

#endif
```

Spreadsheet.cpp

```
#include "Spreadsheet.h"
#include "AddStrings.h"
#include "Cell.h"
#include "DynamicCell.h"
#include <iostream>

bool Spreadsheet::freePosition(Coordinates _coord){

    std::string key = _coord.toString();

    if (freeInteger(_coord)) {
        if (freeString(_coord)) {
            if (freeDCell(_coord)) {
                return true;
            }
        }
    }

    return false;
}

template<>
void Spreadsheet::add<String>(String s)
{
    Coordinates _coord = s.getCoordinates();
    std::string key = _coord.toString(); // get key

    if (freePosition(_coord))
        _spreadsheetString.emplace(key, s);
    else
        std::cout << "position " << key << " already occupied" << std::endl;

}


template<>
void Spreadsheet::add<Integer>(Integer i)
{
    Coordinates _coord = i.getCoordinates();
    std::string key = _coord.toString();

    if (freePosition(_coord))
        _spreadsheetInteger.emplace(key, i);
    else
        std::cout << "position " << key << " already occupied" << std::endl;
}

template<>
void Spreadsheet::add<AddStrings>(AddStrings Ds)
{
    Coordinates _coord = Ds.getCoordinates();
    std::string key = _coord.toString(); // get key
    std::string st = Ds.getValue();

    if (freePosition(_coord)){
        String s(_coord.getLine(), _coord.getColumn(), st);
        _spreadsheetString.emplace(key, s);}
    else
        std::cout << "position " << key << " already occupied" << std::endl;

}

std::string Spreadsheet::getContent(int line, int column)
{
    Coordinates _coord(line, column);
    std::string key = _coord.toString();

    if (freePosition(_coord)) {
        return "null: not initialized";
    } else {
        try {
            _spreadsheetInteger.at(key);
```

```
        } catch (const std::out_of_range& e) {
            return _spreadsheetString.at(key).getValue();
        }
        return std::to_string(_spreadsheetInteger.at(key).getValue());
    }
}

void Spreadsheet::print() {

    std::cout << this;

}

std::ostream& operator<<(std::ostream& os, Spreadsheet obj)
{
    os << obj.toString();
    return os;
}

std::string Spreadsheet::toString(){
    std::string res = "";
    res += "This spreadsheet contains:\n";
    for ( auto it = _spreadsheetString.begin(); it != _spreadsheetString.end(); ++it )
        res += " " + it->first + ": " + (it->second).getValue() + "\n";
    for ( auto it = _spreadsheetInteger.begin(); it != _spreadsheetInteger.end(); ++it
)
        res += " " + it->first + ": " + std::to_string((it->second).getValue()) +
"\n";

    return res;
}

bool Spreadsheet::freeString(Coordinates _coord){

    std::string key = _coord.toString();

    try {
        _spreadsheetString.at(key);
    } catch (const std::out_of_range& e) {
        return true;
    }
    return false;
}

bool Spreadsheet::freeInteger(Coordinates _coord){

    std::string key = _coord.toString();

    try {
        _spreadsheetInteger.at(key);
    } catch (const std::out_of_range& e) {
        return true;
    }
    return false;
}

bool Spreadsheet::freeDCell(Coordinates _coord){

    std::string key = _coord.toString();

    try {
        _spreadsheetDCell.at(key);
    } catch (const std::out_of_range& e) {
        return true;
    }
    return false;
}
```

```
int main() {

    Spreadsheet s;

    String s1(1,2,"hello");
    s.add(s1);
    std::cout << s1;

    AddStrings as(1,5,String(1,12,"hello "), String(1,5,"world"));
    std::cout << as.getValue() << std::endl;


    s.add(Integer(1,2,42));
    s.add(Integer(1,5,2));
    s.add(String(1,5,"world"));

    s.add(as);

    if (Integer(1,2,42)<Integer(1,4,50)) {
        std::cout << "its working" << std::endl;
    }

    std::cout << s;

    std::cout << s.getContent(1,2) << std::endl;
    std::cout << s.getContent(1,5) << std::endl;
    std::cout << s.getContent(1,6) << std::endl;
}
```

String.cpp

```
#include "String.h"

String::String(int line, int column, std::string s): Content(line,column,s) {
setAsString();}

std::ostream& operator<<(std::ostream& os, String& dt)
{
    os << dt.getCoordinates().toString() << " " << dt.getValue() << std::endl;
    return os;
}
```

String.h

```
#ifndef __STRING_H__
#define __STRING_H__


#include "Content.h"

class String: public Content<std::string>{

public:

    String(int line, int column, std::string s);

    friend std::ostream& operator<<(std::ostream& os, String& dt);

};

#endif
```