Date received:

SCHOOL OF ENGINEERING

COVER SHEET FOR CONTINUOUSLY ASSESSED WORK

Course Code ……EE40GA……

SECTION 1: **Student to complete**

SURNAME/FAMILY NAME: ……OSTERMAN……..

FIRST NAME: ……BRIAN……..

ID Number: ……51337831…….

Date submitted: ……21/11/2016…….

Please:

- *Read the statement on "Cheating" and definition of "Plagiarism" contained over page. The full Code of Practice on Student Discipline, Appendix 5.15 of the Academic Quality Handbook is at: www.abdn.ac.uk/registry/quality/appendices.shtml#section5*

- *attach this Cover Sheet, completed and signed to the work being submitted*

SECTION 2: Confirmation of Authorship

The acceptance of your work is subject to your signature on the following declaration:

I confirm that I have read, understood and will abide by the University statement on cheating and plagiarism defined over the page and that this submitted work is my own and where the work of others is used it is clearly identified and referenced. I understand that the School of Engineering reserves the right to use this submitted work in the detection of plagiarism.

Signed: _____BRIAN JACK OSTERMAN_____

Date:_____15/11/2016_____

Note: Work submitted for continuous assessment will not be marked without a completed Cover Sheet. Such work will be deemed 'late' until a completed cover Sheet is submitted and will be subject to the published penalty for late submission.

Cheating in any assessment, whether formative or summative, can result in disciplinary action being taken under the University's Code of Practice on Student Discipline. For these purposes "Cheating" includes:

(a) Possession in an examination of material or electronic device which has not been authorised in writing by the relevant Course Co-ordinator. Students whose first language is not English may, however, refer to a dictionary where this is approved by the Head of the School responsible for the examination;

(b) Copying from another student in an examination;

(c) Removing an examination book from an examination room;

(d) Impersonating another candidate in relation to any assessment;

(e) Permitting another person to impersonate oneself in relation to any assessment;

(f) Paying or otherwise rewarding another person for writing or preparing work to be Submitted for assessment;

(g) Colluding with another person in the preparation or submission of work which is to be assessed. This does not apply to collaborative work authorised by the relevant course coordinator.

(h) Plagiarism. Plagiarism is the use, without adequate acknowledgment, of the intellectual work of another person in work submitted for assessment. A student cannot be found to have committed plagiarism where it can be shown that the student has taken all reasonable care to avoid representing the work of others as his or her own.

EE40GA Computer and Software Engineering
CA: Sorting Program

Written By Brian J. Osterman
Group Members: James Short, James MacLean

Supervisor: Dr. Fabio Verdicchio

51337831
brian.jack.osterman.13@aberdeen.ac.uk

Contents

1. Introduction

This design and development project sets out to deliver a sorting program by a specific deadline. The idea behind it is to work program through with proper project management guidelines and layouts, effectively managing time and delivering individual releases of the program to internal deadlines set by the individual group. The necessities of the program are broken down into 3 releases, with a 4th as optional to produce if time allows. While each release has its own development cycle, certain components of releases may overlap and thus for optimal time management end up being developed concurrently across stages.
An object-oriented approach to the program is used as objects can easily be passed and reused across files, and intricate knowledge of the object itself is not needed for its implementation in files and programs.

2. Project Requirements

The project has separate releases for incremental development. Each release covers a different set of tasks ultimately ending with the full development and integration of the system.

2.1 Release 1

The first release considers a simple item, comprising one string storing a name (optionally two strings, name and family name; now this becomes a composite item, see next); and a composite item Date of Birth (DoB), comprising day, month, and year.

Input Values can be:

- Generated randomly by dedicated functions of the program.
- Supplied via keyboard.

Output values:

- Printed to screen

2.2 Release 2

The second release considers the following composite item:

- First name, Family name, Bio-data

2.3 Release 3

Support a studentrecord_item, comprising the following:

- First name, Middle name, Family name;
- Student ID, Nationality (string), Email Address (string).
- DoB, Bio-data.

2.4 Waterfall Development

The waterfall development method is a progressive development cycle for software in which each task is broken up to be written incrementally. Each individual section should be working to its intended purpose before being integrated into the system. This is ensured by testing the system for each step so that the final product is in full working order for system integration.

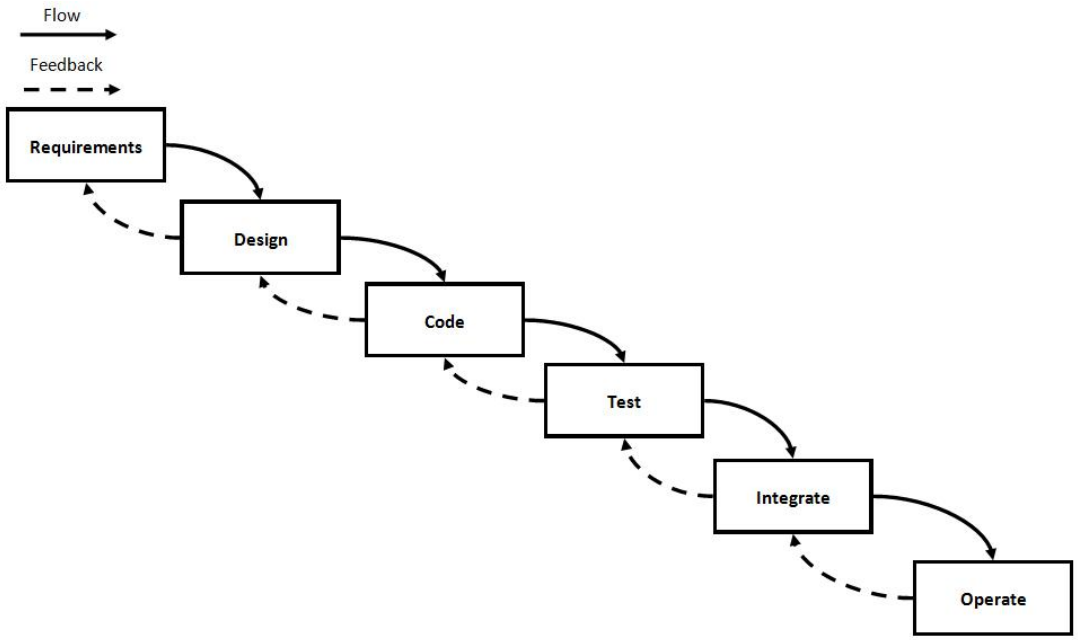

Figure 1: Waterfall Development Diagram

3. Development Plan

3.1 Method

Since there are multiple people working on the project, tasks can be divided and worked on simultaneously. As such the development cycle is broken down and tasks are allocated to different users. This alleviates time needed to develop the system.

3.2 Task Breakdown

i. Release 1

Firstly for release one the team must collaborate on how to approach the individual design tasks. This is to be done for each release cycle as it is important for each member and the group as a whole to have a plan of attack for the development.

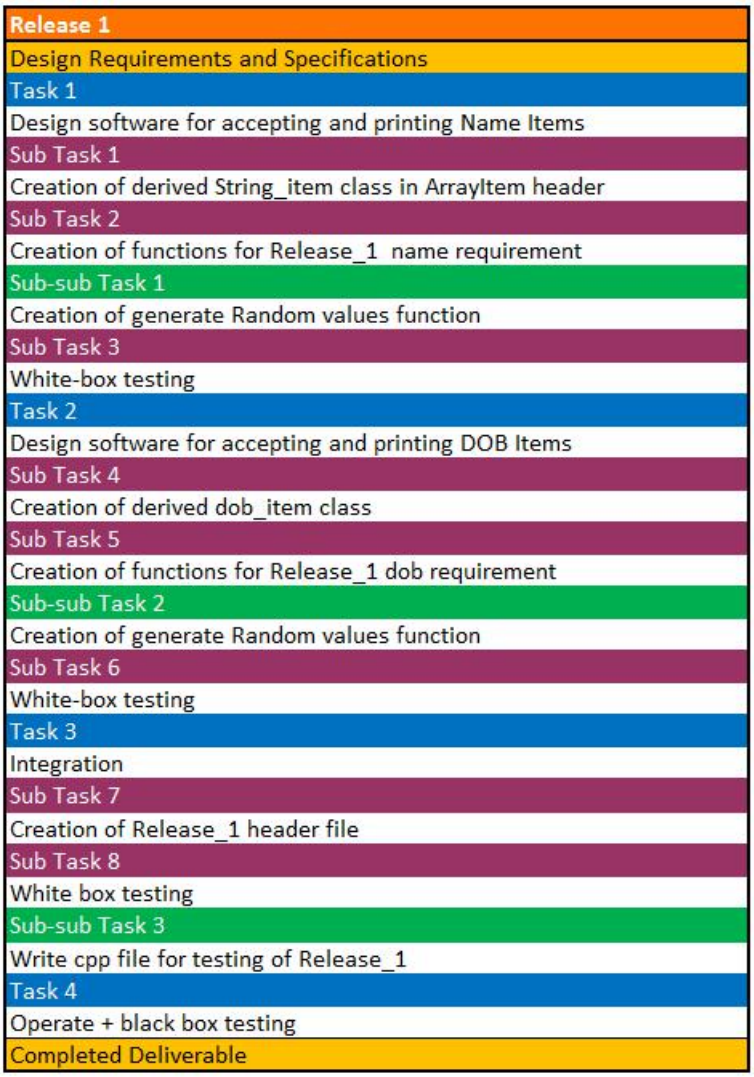

Figure 2: Release 1 Task Breakdown

The first release calls for the development of two tasks: the name_item and date_of_birth_item tasks. These two tasks are of different item types. With name_item being a string, and date_of_birth_item being numeric type (int, double, float). In the Array_item header file a class is constructed derived from the basic_item class. This class contains functions to take strings entered to the application and assign them to string_object. This class is called string_item and it can be passed to functions of other class members and even reused in subsequent releases.

The second task is produced in parallel with the first and it does the same as the first task, except that the difference is that instead of using strings this class uses int, double, or float type values.

For each task one team member was focused on developing functions to generate random items. This function is utilized across all releases in the project. After these

tasks are completed, they are integrated into the system through testing in order to ensure that the system performs as needed.

ii. Release 2

As with the first release this one is also broken down into different components for a more efficient development process. Since there is already a names class it can be integrated into this release, reducing the amount of work. The biodata class is then created in a header file with functions that use objects of the integer_item class from Array_item header. Since the objects for biodata use the integer_items, time can be directed towards function verification instead of writing new functions.

Release 2
Design Requirements and Specifications
Task 1
Design software for accepting and printing BIODATA items
Sub-Task 1
Creation of derived biodata_item class from basic_item class
Sub-sub Task 1
Creation of age weigh height objects
Sub Task 2
Creation of functions for Release_2 biodata requirement
Sub-sub Task 2
Creation of generate Random values function
Sub Task 3
White-box testing
Task 2
Design software for accepting and printing name Items
Sub Task 4
Reuse class from Release 1
Task 3
Integration
Sub Task 5
Creation of functions for Release_2 header file
Sub Task 6
White-box testing
Sub-sub Task 3
Write cpp file for testing of Release_2
Task 4
Operate + black box testing
Completed Deliverable

Figure 3: Release 2 Task Breakdown

iii. Release 3

For this release same principles for the previous two apply. Release 3 should be the fastest to complete because of the object oriented approach and the functions developed previously. Reading and writing from a file is an optional component, and therefore the last task is only dependent on the first. Since the requirements of this

release are ultimately complex, the longest part of this release is its testing and implementation into the program.

Release 3
Design Requirements and Specifications
Task 1
Design software for accepting and printing student record items
Sub-Task 1
Creation of derived student_record_item class
Sub-sub Task 1
Creation of necessary objects
Sub Task 2
Creation of functions for Release_3 requirement
Sub-sub Task 2
Creation of generate Random values function
Sub Task 3
White-box testing
Task 2
Create function to read from text file
Task 3
Integration
Sub Task 4
White-box testing
Sub-sub Task 3
Write cpp file for testing of Release_3
Task 4
Operate + black box testing
Completed Deliverable

Figure 4: Release 3 Task Breakdown

3. 3 Production Plan

The below Gantt chart displays how the production cycle for each individual task occurs as the project deadline approaches. The first two tasks for each release are always worked on at the same time. Tasks 3 and 4 are dependent on the previous tasks, and as such can only be completed after the previous tasks are complete. Since the random generation and testing for each task are necessary to the completion of each release they last 1/3 of the time spent for each cycle. As there are 3 releases each one is divided into 7 day development periods.

While each individual release is being tested, the other two members of the team move on to the next task. This shortens the development cycle and reduces time needed overall for each segment.

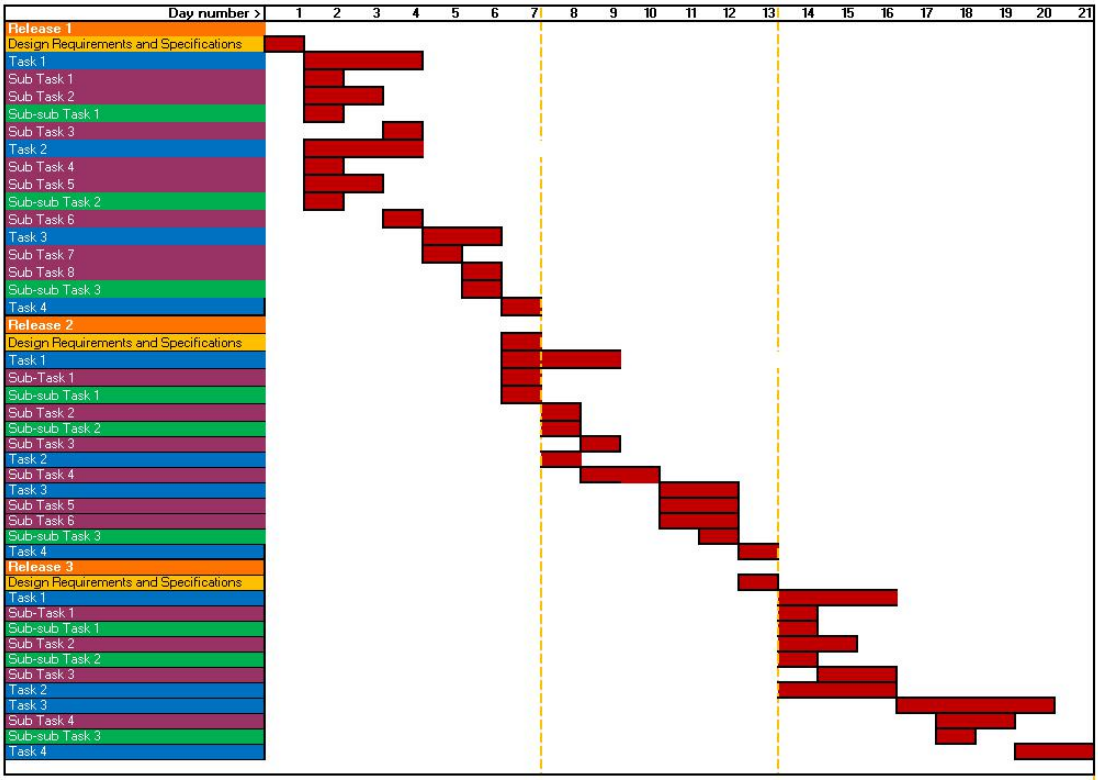

Figure 5: Gantt Chart of Development Timeline

3. 4 Release Diagram

The below release diagram only shows the main tasks for each release and how they coincide with the rest of the progressive development. This diagram provides a clearer depiction of how the simultaneous development works to reduce time needed for each release.

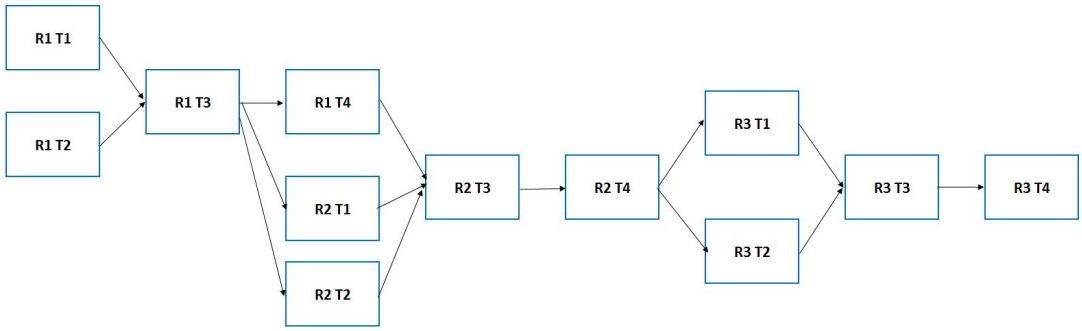

Figure 6: Network Release Diagram

3. 5 Task Distribution

Once the project specifications are broken down into different tasks they are then allocated to different developers. This gives a clear projection of what each developer needs to accomplish, ensuring that workout is distributed appropriately.

i. Release 1

TASK	Responsiblility of
Release 1	
Design Requirements and Specifications	ALL
Task 1	JS
Sub Task 1	JS
Sub Task 2	JS
Sub-sub Task 1	JM
Sub Task 3	JM + JS
Task 2	BO
Sub Task 4	BO
Sub Task 5	BO
Sub-sub Task 2	JM
Sub Task 6	JM + BO
Task 3	ALL
Sub Task 7	JM
Sub Task 8	ALL
Sub-sub Task 3	JM
Task 4	BO + JM

Figure 7: Release 1 Task Allocation

ii. Release 2

TASK	Responsiblility of
Release 2	
Design Requirements and Specifications	ALL
Task 1	JS
Sub-Task 1	JS
Sub-sub Task 1	JS
Sub Task 2	JS
Sub-sub Task 2	JM
Sub Task 3	JM
Task 2	BO
Sub Task 4	BO
Task 3	ALL
Sub Task 5	BO
Sub Task 6	BO
Sub-sub Task 3	BO
Task 4	JM

Figure 8: Release 2 Task Allocation

iii. Release 3

TASK	Responsibility of
Release 3	
Design Requirements and Specifications	ALL
Task 1	JS
Sub-Task 1	JS
Sub-sub Task 1	JS
Sub Task 2	JS
Sub-sub Task 2	JM
Sub Task 3	JM
Task 2	BO
Task 3	ALL
Sub Task 4	BO
Sub-sub Task 3	BO
Task 4	ALL

Figure 9: Release 3 Task Allocation

4. Block Diagrams and Dependencies

4. 1 Release 1

i. Block Diagram

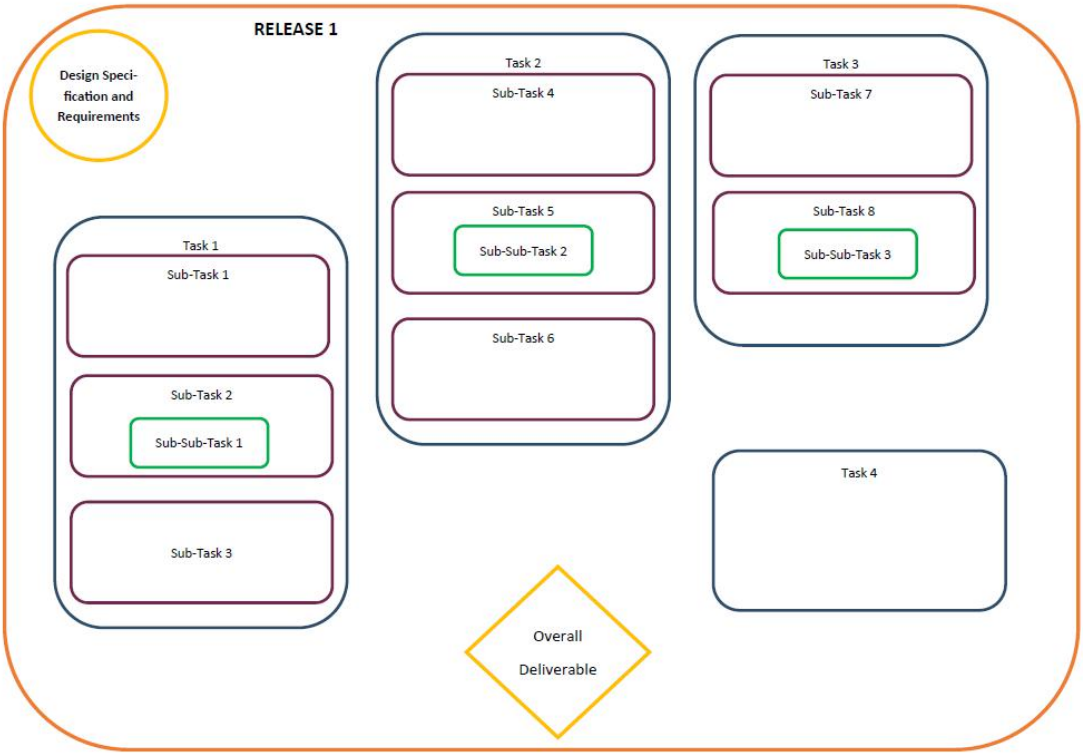

Figure 10: Release 1 Block Diagram

ii. Release Depedencies

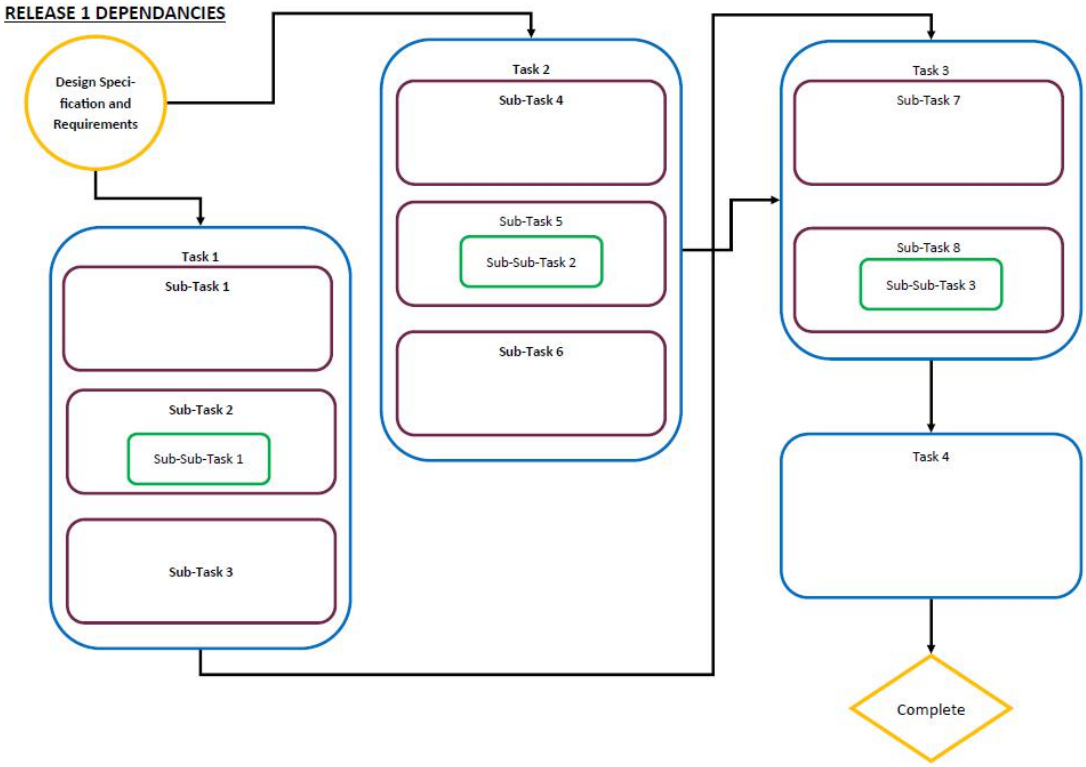

Figure 11: Release 1 Task Dependencies

4. 2 Release 2

i. Block Diagram

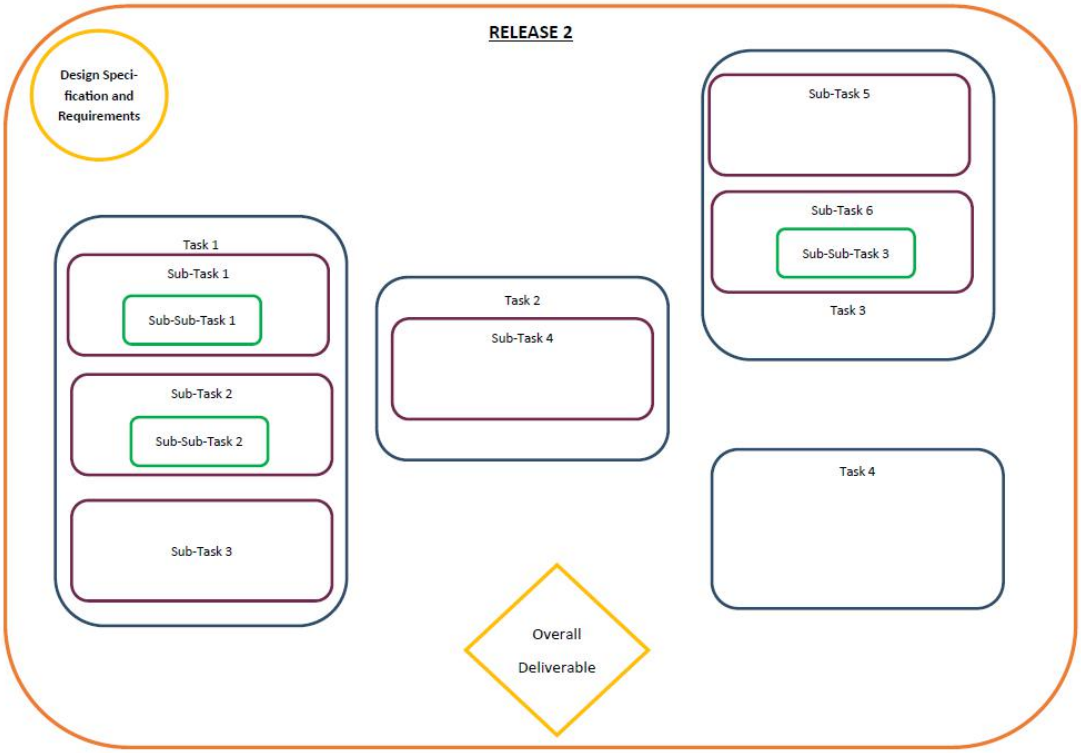

Figure 12: Release 2 Block Diagram

ii. Release Dependencies

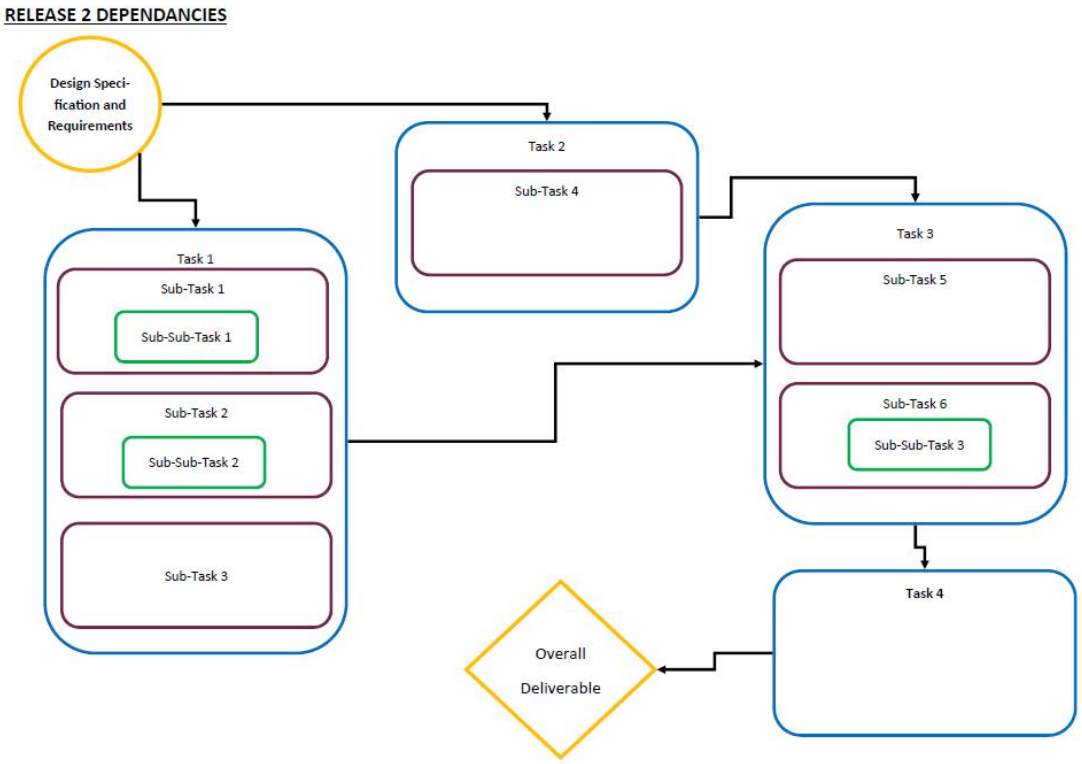

Figure 13: Release 2 Task Dependencies

4. 3 Release 3

i. Block Diagram

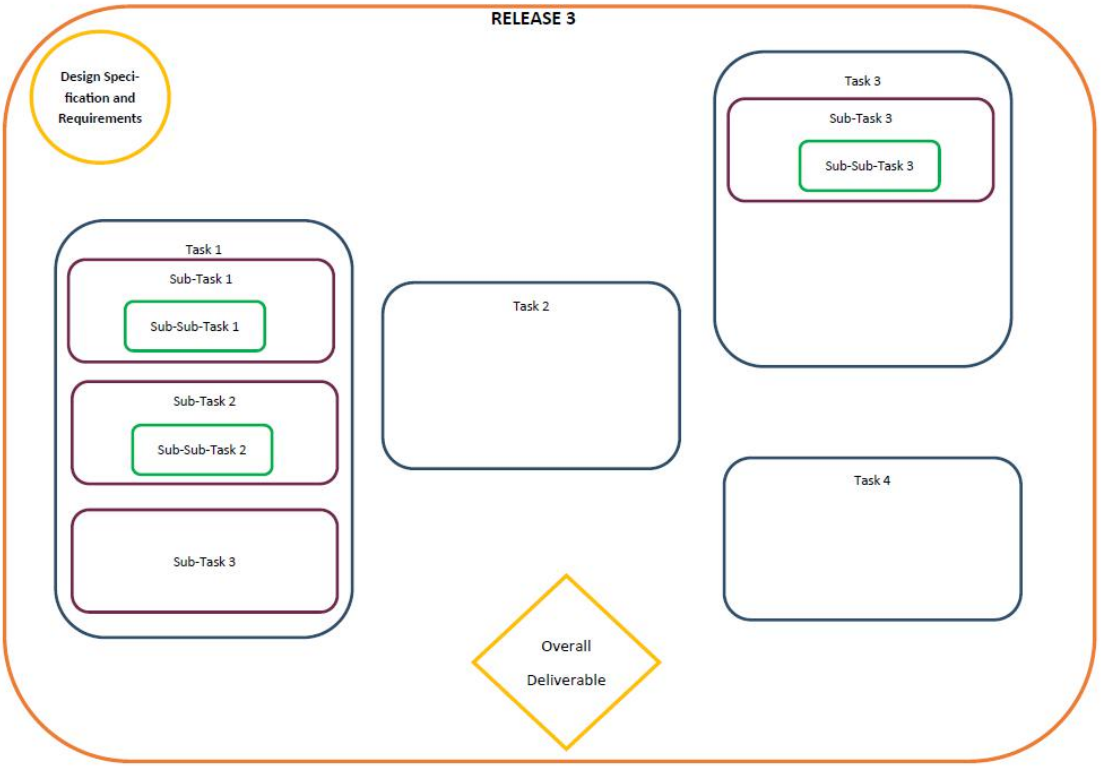

Figure 14: Release 3 Block Diagram

ii. Release Dependencies

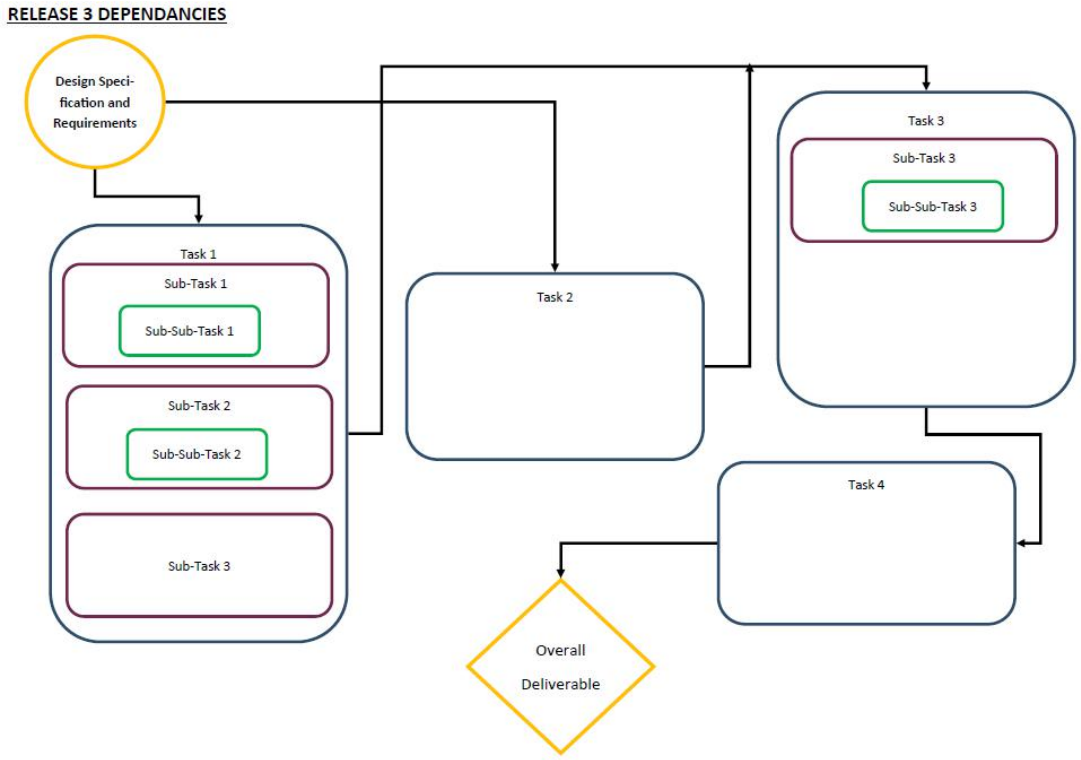

Figure 15: Release 3 Task Dependencies

4. 4 Overall System

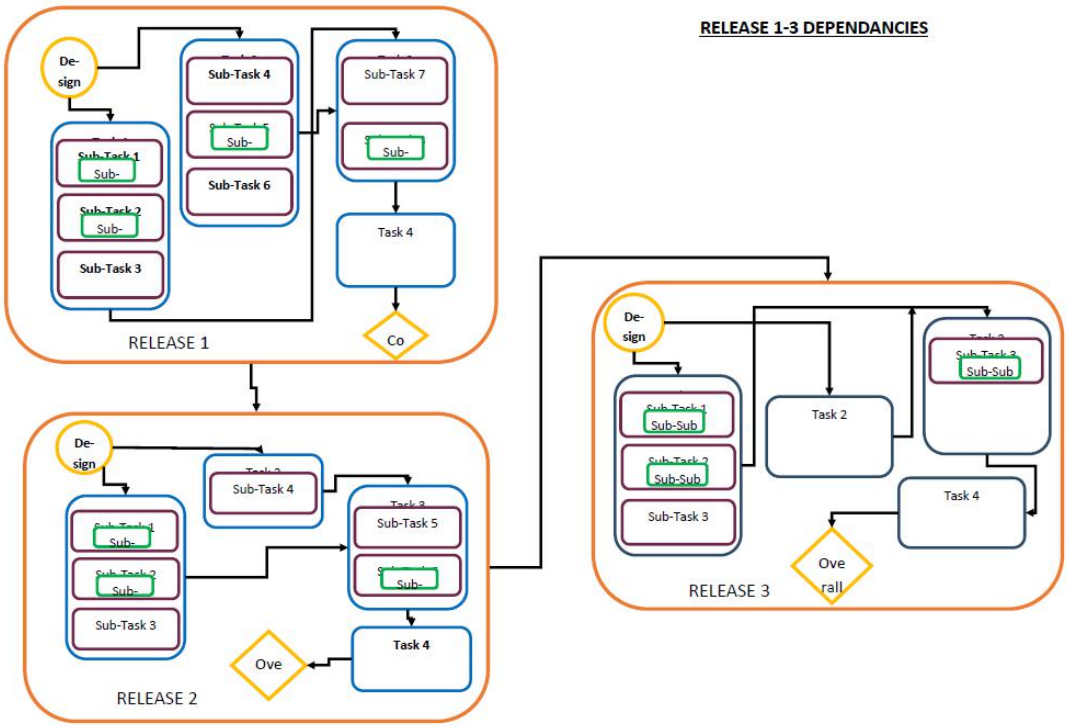

Figure 16: Overall system Dependencies

5. Implementation

5. 1 Release 1

```
1       class stringitem : public basic_item{
2
3
4       public:
5               stringitem(){ ; }
6               stringitem(){ ; }
7               string stringobj;
8
9
10              virtual void printItemOnScreen()
11              {
12                      cout << "You entered " << stringobj << "." << endl;
13              }
14              virtual void enterItemFromKeyboard()
15              {
16                      getline(cin, stringobj);
17              }
18
19              virtual void generateRandomItem() { ; }
20              virtual bool IsLargerThan(basic_item* other_item, basic_sort_criteria*
                sort_criteria = NULL)
21              {
22                      return false;
23              }
24      };
```

For each item that requires a string this class can be used because of the use of object-oriented programming. (names, email addresses, nationalities; All are string types)

i. Main Functions

Objects first_name and family_name are of string_item and therefore accept strings as inputs. A similar flowchart to the one below for all string type items used throughout the project.

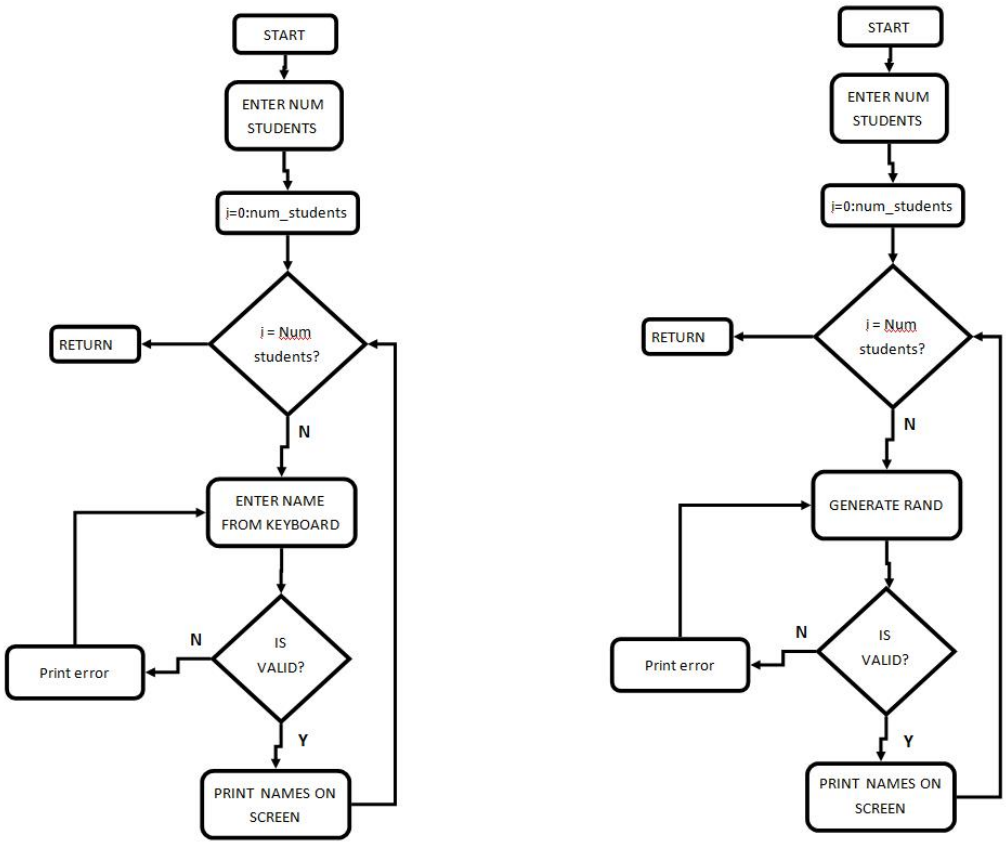

Figure 17: name_item function flowchart

And for Date of Birth the same applies. Other functions and classes that require integer types have a similar flowchart, adjusted with respect to its specific use:

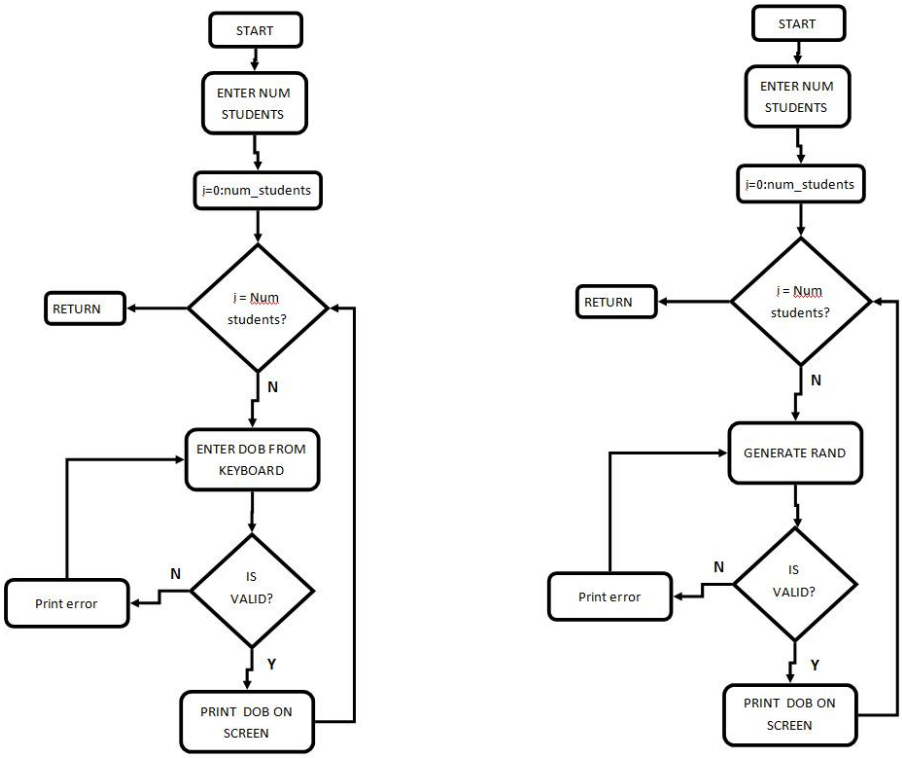

Figure 18: Date of birth function flowchart

ii. Integration

For the program itself all functions and classes are finally integrated into a .cpp source file for execution.

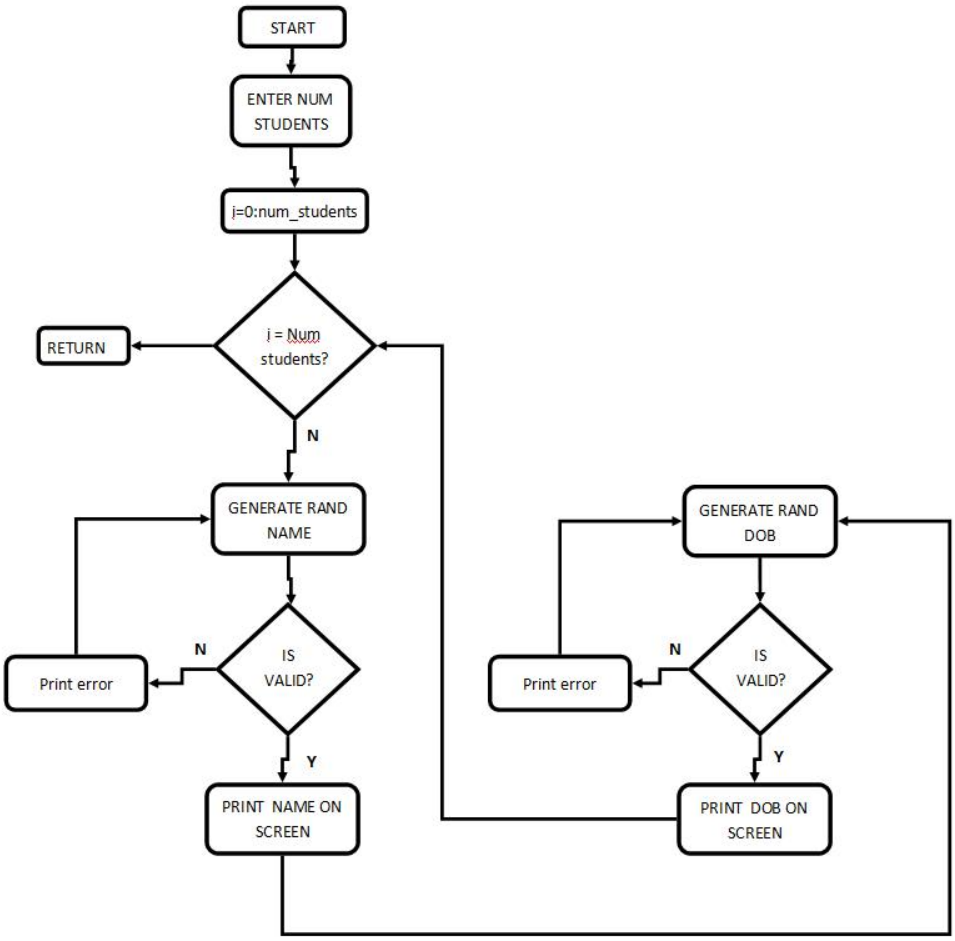

Figure 19: Release 1 Flowchart

It is shown in figure 19 above the logic path that the program follows in execution.

5.2 Release 2

Release two builds on the blocks from release one, by using the classes created to derive classes for the last name (family name) and the biodata. The have the same logic flowcharts as their respective item types from the first release (string for family name, integer for biodata). These are then integrated using a .cpp file for compilation and execution.

5.3 Release 3

As with the previous two builds, the third release has classes developed based on their respective item types (strings, integers) and then integrated into the systems .cpp file for full execution of the studentrecord_item. This makes an object that contains all elements of the name, student record, and email address.

Optionally, once these three releases are completed the group may or may not decide to work on a merge-sort sorting algorithm. This splits the array so that less iterations of sorting need to occur in order for an array to be properly sorted. Thus reducing time and memory usage.

6. Conclusion

The software development cycle is similar to that of other development cycles. The main difference with it when compared to other types is that it relies on constant testing and adjusting. This can in some cases be more time consuming, but ultimately ensures that the product works as intended throughout as much of the development as possible. Managing the development of software requires the same planning and workload distribution as any other type of development and as such the same internal deadlines and project completion date are set in accordance with team decisions.

7. Code Appendix

```
#ifndef RELEASE_1_H
#define RELEASE_1_H

#include "ArrayItem.h"
#include "sort.h"
#include <string>
#include "time.h"


class names : public basic_item{
protected:

	stringitem first_name;
	stringitem family_name;
public:
    names(){;}
	~names(){ ; };


	virtual void printItemOnScreen(){

		if (isEmpty()){
			cout << "! Blank ! " << endl;
		}
		else{
			cout << "First name is: " << endl;
			first_name.printItemOnScreen();
			cout << "Family name name is: " << endl;
			family_name.printItemOnScreen();
		}
	}

	virtual void enterItemFromKeyboard()
	{
		cout << "Insert first name : " << endl;
		first_name.enterItemFromKeyboard();

		cout << "Insert family naem : " << endl;
		family_name.enterItemFromKeyboard();
```

```
            // item filled
            empty = false;

        }

    virtual bool IsLargerThan(basic_item* other_item, basic_sort_criteria*
sort_criteria=NULL){return false;}

    ////////////////////// Gen Rand It ////////////////////////////////

    virtual void generateRandomItem(){

           first_name.generateRandomName();
           family_name.generateRandomName();

           }


    virtual basic_item* allocateNewItem(){ ; }
    virtual void deleteOtherItem(basic_item* item){ ; }
};

class dob_item : public basic_item{

protected:
    int day,month,year;

       public:
    dob_item(){;}
    ~dob_item(){;}

    int getDay(){
        return day;
    }
    int getMonth(){
        return month;
    }
    int getYear(){
        return year;
    }
    void printItemOnScreen(){
        if (isEmpty()){
            cout << "Item isn't there."<<endl;
        }
        else{
            cout << "Date of Birth: " << day << "/" << month <<"/" << year << endl;
        }
    }
    void enterItemFromKeyboard(){
        cout << "Enter day of birth:" << endl;
        cin >> day;
        cout << "Enter month of birth:" << endl;
        cin >> month;
        cout << "Enter year of birth:" << endl;
        cin >> year;

        empty = false;
    }
    bool date_check(){
        int max_days;

        if (month<1 || month>12){
            cout << "Month range exceeded. Start again." << endl;
```

```
            enterItemFromKeyboard();
            return false;
        }
        if (year < 0){
            cout << "This is before Jesus' time. Enter a proper date will you?" <<
endl;
            enterItemFromKeyboard();
            return false;
        }
        switch (month) {
            case 1:
            case 3:
            case 5:
            case 7:
            case 8:
            case 10:
            case 12:
                max_days = 31;
                break;
            case 4:
            case 6:
            case 9:
            case 11:
                max_days = 30;
                break;
            case 2:
                if ((year % 4 == 0) || (year % 400 == 0))
                    max_days = 29;
                else if(year % 100 == 0)
                    max_days = 28;
                else
                    max_days = 28;
                break;
            default:
                return false;
        }
        if ((day<1) || (day>max_days)){
            cout << "Day range exceeded. Start again." << endl;
            enterItemFromKeyboard();
            return false;
        }
        else
            return true;
    }
    bool IsLargerThan(basic_item* other_item, basic_sort_criteria*
sort_criteria=NULL){
        bool result=false;

        // if the other item is "empty" (non allocated) don't do any comparison
        if(other_item==NULL)
            return false;


        // first typecast the other item to confimr it is the same as this;
        dob_item* typecasted_other_item = typecastItem(other_item, this);

        // check that it worked
        if(typecasted_other_item==NULL)
        {
            cout << "Other item is not of type integer_item." << endl;
            return false;
            // items of the wrong type (or null pointers) will be pushed to the end of
the list
        }
```

```
        // now verify if the other item is larger than the curren
        if( getDay() > (typecasted_other_item->getDay()))
            result=true;

        // chek if there are sorting options to apply
        if(sort_criteria!=NULL)
        {
            // if sorting is in descending order the result is reversed
            if( !( sort_criteria->getAscending() ) )
                result=!result;
        }

        return result;
    }
   // void generateRandomDOB(){

   // }
        virtual basic_item* allocateNewItem(){ return NULL; }
        virtual void deleteOtherItem(basic_item* item){ ; }

        virtual void generateRandomItem(){
               int days, months, years;
               days = rand() % (30 + 1 - 1) + 1;
               months = rand() % (12 + 1 - 1) + 1;
               years = rand() % (3000 + 1 - 0) + 0;

               day = days;
               month = months;
               year = years;

               empty = false;
        }

};


#endif /* Release_1_h */
```

```
#ifndef RELEASE_2_H
#define RELEASE_2_H

#include "Release_1.h"


class bio_item : public basic_item{
public:
       integer_item age, height, weight;

public:
       bio_item(){ ; }
       ~bio_item(){ ; }



       virtual void printItemOnScreen()
       {
              if (isEmpty())
                     cout << "Item is empty." << endl;
              else
              {
                     cout << "age is ";
                     age.printItemOnScreen();
                     cout << endl;
```

```
                cout << "weight is ";
                weight.printItemOnScreen();
                cout << endl;

                cout << "height is ";
                height.printItemOnScreen();
                cout << endl;
        }
    }

    virtual void enterItemFromKeyboard()
    {
        cout << "Insert age ." << endl;
        age.enterItemFromKeyboard();

        cout << "Insert weight ." << endl;
        weight.enterItemFromKeyboard();

        cout << "Insert height ." << endl;
        height.enterItemFromKeyboard();

        // item filled
        empty = false;
        //empty=( age.isEmpty() || weight.isEmpty() || height.isEmpty() );
    }

    virtual bool  vailidity_check()
    {
        if ((age.getItemVal() > 0) && (weight.getItemVal() > 0) &&
(height.getItemVal() > 0))
        {
            return true;
        }
        else { return false; }
    }

    virtual void biodata_program()
    {
        enterItemFromKeyboard();
        if (vailidity_check())
        {
            printItemOnScreen();
        }
        else
        {
            cout << "invalid data entry - non-zero, positive integers only"
<< endl;
            enterItemFromKeyboard();
        }
    }

    virtual void generateRandomBioData()
    {
        age.generateRandomAge();
        weight.generateRandomWeight();
        height.generateRandomHeight();
    }

    virtual void execute_biodata()
    {
        int number_of_student = 7, i;

        for (i = 0; i < number_of_student; i++)
        {
            generateRandomBioData();
```

```
                        if (vailidity_check())
                        {
                                printItemOnScreen();
                        }
                        else cout << "invalid data given" << endl;
                }
        }



        //These must be implemented by any derived item

        //virtual void loadItemFromFile(FILE* fin)=0;

        // add anotehr function: IsEqualTo(...)

        virtual basic_item* allocateNewItem(){ return NULL; }
        virtual void deleteOtherItem(basic_item* item){ ; }

        virtual bool IsLargerThan(basic_item* other_item, basic_sort_criteria*
sort_criteria = NULL)
        {
                /*bool result = false;

                // if the other item is "empty" (non allocated) don't do any comparison
                if (other_item == NULL)
                        return false;


                // first typecast the other item to confimr it is the same as this;
                bio_item* typecasted_other_item = typecastItem(other_item, this);

                // check that it worked
                if (typecasted_other_item == NULL)
                {
                        cout << "Other item is not of type integer_item." << endl;
                        return false;
                        // items of the wrong type (or null pointers) will be pushed to
the end of the list
                }

                // now verify if the other item is larger than the curren
                if (getItemVal() > (typecasted_other_item->getItemVal()))
                        result = true;


                // chek if there are sorting options to apply
                if (sort_criteria != NULL)
                {
                        // if sorting is in descending order the result is reversed
                        if (!(sort_criteria->getAscending()))
                                result = !result;
                }

                return result;*/ return NULL;
        }

};


#endif /* Release_2_h */
```

```
#ifndef RELEASE_3_H
```

```
#define RELEASE_3_H


#include "Release_2.h"
#include "ArrayItem.h"


class studentrecord_item : public basic_item
{

public:

	stringitem first_name;
	stringitem middle_name;
	stringitem family_name;
	integer_item student_id;
	stringitem email;
	stringitem nationality;
	dob_item dob;
	biodata_item  bio;

	virtual void enterItemFromKeyboard()
	{
		cout << "Insert first name ." << endl;
		first_name.enterItemFromKeyboard();

		cout << "Insert middle name ." << endl;
		middle_name.enterItemFromKeyboard();

		cout << "Insert family name ." << endl;
		family_name.enterItemFromKeyboard();

		cout << "insert student I.D number" << endl;
		student_id.enterItemFromKeyboard();

		cout << "Enter Nationality" << endl;
		nationality.enterItemFromKeyboard();

		cout << "Enter EMAIL address" << endl;
		email.enterItemFromKeyboard();

		// item filled
		empty = false;

	}

	virtual void enterItemFromRandom()
	{
		//cout << "Insert first name ." << endl;
		first_name.generateRandomName();

		//cout << "Insert middle name ." << endl;
		middle_name.generateRandomName();

		//cout << "Insert family name ." << endl;
		family_name.generateRandomName();

		//cout << "insert student I.D number" << endl;
		student_id.generateRandomItem();

		//cout << "Enter Nationality" << endl;
		nationality.generateRandomNationality();

		//cout << "Enter EMAIL address" << endl;
		//email.generateRandomEmail;
```

```
		// item filled
		empty = false;

	}

	virtual void printItemOnScreen()
	{
		if (isEmpty())
			cout << "Item is empty." << endl;
		else
		{
			cout << "first name is ";
			first_name.printItemOnScreen();
			cout << endl;

			cout << "middle name is ";
			middle_name.printItemOnScreen();
			cout << endl;

			cout << "Family name is ";
			family_name.printItemOnScreen();
			cout << endl;

			cout << "Student I.D  is ";
			student_id.printItemOnScreen();
			cout << endl;

			cout << "Nationality  is ";
			nationality.printItemOnScreen();
			cout << endl;

			cout << "EMAIL address  is ";
			email.printItemOnScreen();
			cout << endl;
		}
	}

	virtual void release_3_program()
	{
		//enterItemFromKeyboard();
		enterItemFromRandom();
		printItemOnScreen();

	};

	virtual void generateRandomItem(){ ; }

	virtual bool IsLargerThan(basic_item* other_item, basic_sort_criteria*
sort_criteria = NULL) {  return NULL; }

	// add anotehr function: IsEqualTo(...)

	// void generateRandomDOB(){

	// }
	virtual basic_item* allocateNewItem(){ return NULL; }
	virtual void deleteOtherItem(basic_item* item){ ; }
};
#endif /* Release_1_h */
```