Due date: Apr 26, 11:59 PM

Homework 4

1. Implement a class IntArr using dynamic memory.

a. data members:
capacity: maximum number of elements in the array
size: current number of elements in the array
array: a pointer to a dynamic array of integers

b. constructors:
default constructor: capacity and size are 0, array pointer is nullptr
user constructor: create a dynamic array of the specified size

c. overloaded operators:
subscript operator: return an element or exits if illegal index

d. “the big three”:
copy constructor: construct an IntArr object using deep copy
assignment overload operator: deep copy from one object to another
destructor: destroys an object without creating a memory leak

e. grow function: “grow” the array to twice its capacity

f. push_back function: add a new integer to the end of the array

g. pop_back function: remove the last element in the array

h. getSize function: return the current size of the array (not capacity)

i. Use the provided main function (next page) and output example

Notes

a. grow function replaces the existing array with a new array

b. subscript operator should be a const member function

c. ensure that deep copy is used where applicable

d. push_back and pop_back functions require the grow function

e. push_back and pop_back functions should update array size (not capacity)

Due date: Apr 26, 11:59 PM

main:

```
int main() {
    cout << endl;

    IntArr a{5};
    for(int i=0; i<5; i++) { a.push_back((i+1)*5); }
    cout << "Array size: " << a.getSize() << endl;

    cout << "Array A: ";
    for(int i=0; i<a.getSize(); i++) {  cout << a[i] << " "; }
    cout << endl << endl;

    a.push_back(30);
    a.push_back(35);
    cout << "Array size: " << a.getSize() << endl;

    IntArr b = a;
    cout << "Array A: ";
    for(int i=0; i<a.getSize(); i++) {  cout << a[i] << " "; }
    cout << "\nPArray B: ";
    for(int i=0; i<b.getSize(); i++) {  cout << b[i] << " "; }
    cout << endl << endl;

    a.pop_back();
    cout << "Array size: " << a.getSize() << endl;
    b = a;
    cout << "Array A: ";
    for(int i=0; i<a.getSize(); i++) {  cout << a[i] << " "; }
    cout << "\nArray B: ";
    for(int i=0; i<b.getSize(); i++) {  cout << b[i] << " "; }
    cout << endl << endl;

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

Output:

```
Array size: 5
Array A: 5 10 15 20 25

Array size: 7
Array A: 5 10 15 20 25 30 35
PArray B: 5 10 15 20 25 30 35

Array size: 6
Array A: 5 10 15 20 25 30
Array B: 5 10 15 20 25 30
```

Due date: Apr 26, 11:59 PM

Use following main function to test your program.

```
int main() {
    cout << endl;

    IntArr a{5};
    for(int i=0; i<5; i++) { a.push_back((i+1)*5); }
    cout << "Array size: " << a.getSize() << endl;

    cout << "Array A: ";
    for(int i=0; i<a.getSize(); i++) {  cout << a[i] << " "; }
    cout << endl << endl;

    a.push_back(30);
    a.push_back(35);
    cout << "Array size: " << a.getSize() << endl;

    IntArr b = a;
    cout << "Array A: ";
    for(int i=0; i<a.getSize(); i++) {  cout << a[i] << " "; }
    cout << "\nPArray B: ";
    for(int i=0; i<b.getSize(); i++) {  cout << b[i] << " "; }
    cout << endl << endl;

    a.pop_back();
    cout << "Array size: " << a.getSize() << endl;
    b = a;
    cout << "Array A: ";
    for(int i=0; i<a.getSize(); i++) {  cout << a[i] << " "; }
    cout << "\nArray B: ";
    for(int i=0; i<b.getSize(); i++) {  cout << b[i] << " "; }
    cout << endl << endl;

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

2. Implement a Person class as specified:

data members:

- **name** - name of the Person
- **size** - number of family member
- **capacity -** size of dynamic array
- **family** – a partially filled dynamic array of string

functions:

Due date: Apr 26, 11:59 PM

Person(): default constructor set name to “TBD”, and capacity to 3, size to 0, and create family with capacity

Person(string): one argument constructor set name, and capacity to 3, size to 0, and create family with capacity

Person(string, int): two arg constructor set name, and set capacity using int arg, size to 0, and create family with capacity

accessor - an accessor for the name variable
mutator - an mutator for the name variable

add() – add a member to the family
grow() – double the capacity of the array
display() – output the name of the Person and the names of all family members

Implement the **big three** for Person class.

Add message for function call.

Use following main function to test your program.

```
int main() {
   cout << endl;

   Person p("Joe");

   p.add("Sarah");
   p.add("John");
   p.add("Nora");
   p.display();

   p.add("Nora");
   cout << endl;

   Person p2 = p;
   p2.setName("Jack");
   p2.add("Sam");
   p2.display();
   cout << endl;

   Person p3;
   p3 = p2;
   p3.setName("Kate");
   p3.add("Tom");
   p3.display();
```

Due date: Apr 26, 11:59 PM

```
    cout<<"\n\nfinal result:"<<endl;
    p.display();
    p2.display();
    p3.display();

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

Output from the main function above:

```
Name:Joe
Family: Sarah John Nora
Person => Grow function was called

Person => copy constructor was called.
Name:Jack
Family: Sarah John Nora Nora Sam

Person => assignment operator was called.
Name:Kate
Family: Sarah John Nora Nora Sam Tom


final result:
Name:Joe
Family: Sarah John Nora Nora
Name:Jack
Family: Sarah John Nora Nora Sam
Name:Kate
Family: Sarah John Nora Nora Sam Tom

Person => destructor was called.
Person => destructor was called.
Person => destructor was called.
```

Grading policy:

Should submit .cpp file format, other format will be not accepted and assigned 0 point directly.

50% points loss if the program doesn't compile. 50% points for the rest.
If the code compiles and runs, full points if it succeeds for all requirements.