Create a class **SavingAccount** that will be used to instantiate bank account objects.

The class should have the following data members:

- **`AcctNbr`** - The Account Number
  (Between 1000 and 10000. Defaults to **0**)
- **`annualInterestRate`** - The annual interest rate
  (Between 0.0 and 100.0. Defaults to **0.0**)
- **`savingBalance`** - The Account's balance
  (Greater or equal to 0.0 Defaults to **0.0**)

and the following member functions:

- A **`constructor`** with no arguments.
- An **`overloaded constructor`** that takes as arguments, the values to set the data members.
- All needed **`setter`** and **`getter`** functions.
- A member function **`crdTrans`** that takes as an argument a value of type double. This value will be ***added*** to the ***account balance***.
- A member function **`dbtTrans`** that takes as an argument a value of type double. This value will be ***subtracted*** from the ***account balance***.

Write a program to test your class by instantiating object and accessing members through this object.
Include in your program, two functions that takes as argument a reference to an object if your class. One function calculates and returns the annual interest and the second function calculates and returns the monthly interest

Annual interest = balance * rate / 100
Monthly interest = Annual interest / 12

SavAcct.h

```
#ifndef SAVACCT_H
#define SAVACCT_H

class SavingAccount {

private:
    unsigned acctNbr;
    double annualInterestRate;
    double balance;

public:
    SavingAccount();
    SavingAccount(unsigned, double, double);

    void setAccount(unsigned, double, double);
    void debTrans(double amnt);
    void crdTrans(double amnt);
    unsigned getAcctNbr() const;
    double getRate() const;
    double getBalance() const;

    void setAcctNbr(unsigned);
    void setRate(double);
    void setBalance(double);
};

#endif
```

SavAcct.cpp

```
#include <iostream.h>
#include "SavAcct.h"

SavingAccount::SavingAccount() {
    setAccount(0, 0.0, 0.0);
}

SavingAccount::SavingAccount(unsigned nbr,
double amnt, double rate){
    setAccount(nbr, amnt, rate);
}

void SavingAccount::setAccount(unsigned nbr,
                               double amnt,
                               double rate){
    setAcctNbr(nbr);
    setBalance(amnt);
    setRate(rate);
}

void SavingAccount::setAcctNbr(unsigned nbr){
    acctNbr = nbr > 1000 && nbr < 10000 ? nbr: 0;
}

void SavingAccount::setBalance(double amnt){
    balance = amnt > 0.0 ? amnt : 0.0;
}

double SavingAccount::getRate() const {
    return annualInterestRate;
}
```



```
void SavingAccount::setRate(double rate) {
    annualInterestRate =
           (rate > 0.0 && rate <=100) ?
            rate : 0.0;
}

unsigned SavingAccount::getAcctNbr() const {
    return acctNbr;
}

double SavingAccount::getBalance() const {
    return balance;
}

void SavingAccount::debTrans(double amnt) {
    amnt = amnt > balance ? amnt : 0.0;
    balance -= amnt;
}

void SavingAccount::crdTrans(double amnt) {
    amnt = amnt > 0.0 ? amnt : 0.0;
    balance += amnt;
}
```

Prog.cpp

```
#include <iostream.h>
#include <iomanip.h>
#include "SavAcct.h"

void debit(SavingAccount&);
void credit(SavingAccount&);
void monthly(SavingAccount&);
void annual(SavingAccount&);
double calAnlInt(SavingAccount&);
double calMonInt(SavingAccount&);
int getChoice();

void main() {
  SavingAccount acctOne(32123,20000.55, 3.0);
  int choice;

  cout << setiosflags(ios::showpoint |
              ios::fixed) << setprecision(2);
  choice = getChoice();
  while(choice >=0 && choice <=3) {
    switch (choice) {
      case 0:debit(acctOne);
        break;
      case 1:credit(acctOne);
        break;
      case 2:monthly(acctOne);
        break;
      case 3:annual(acctOne);
        break;
    };
    choice = getChoice();
  }
}
```



```
int getChoice() {
  int choice;
  cout <<"\nEnter: \n"
      << "\t 0) Debit Account.\n"
      << "\t 1) Credit Account.\n"
      << "\t 2) Calculate Monthly Interest.\n"
      << "\t 3) Calculate Yearly Interest.\n"
      << "\t 9) Exit" << endl;

  cin >> choice;
  return choice;
}

void debit(SavingAccount &acct) {
  double amnt;

  cout << "Enter amount to debit:  ";
  cin >> amnt;
  acct.debTrans(amnt);
  cout << "Account #: " << acct.getAcctNbr();
  cout << "\nBalance  :$ " << acct.getBalance();
  cout << endl;
}

void credit(SavingAccount &acct) {
  double amnt;

  cout << "Enter amount to credit:  ";
  cin >> amnt;
  acct.crdTrans(amnt);
  cout << "Account #: " << acct.getAcctNbr();
  cout << "\nBalance  :$ " << acct.getBalance();
  cout << endl;
}
```

```
void monthly(SavingAccount &acct) {
  cout << "Account #: " << acct.getAcctNbr();
  cout << "\nBalance  :$ " << acct.getBalance();
  cout << "\nRate     :$ " << acct.getRate();
  cout << "\nInterest :$ " << calMonInt(acct);
  cout << endl;
}
void annual(SavingAccount &acct) {
    cout << "Account #: " << acct.getAcctNbr();
    cout << "\nBalance  :$ " <acct.getBalance();
    cout << "\nRate     :$ " << acct.getRate();
    cout << "\nInterest :$ " << calAnlInt(acct);
    cout << endl;
}

double calAnlInt(SavingAccount & acctRef) {
    return acctRef.getBalance() *
              acctRef.getRate()/ 100.0;
}

double calMonInt(SavingAccount & acctRef) {
    return calAnlInt(acctRef) / 12;
}
```