

C++

Modify your test program by asking the user to enter the number of accounts (object) he needs to create.

Your program should create dynamic objects and manage their use.

SavAcct.h

```
#ifndef SAVACCT_H
#define SAVACCT_H

class SavingAccount {
private:
    unsigned acctNbr;
    double annualInterestRate;
    double balance;
public:
    friend double calAnlInt(SavingAccount&);
    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
```

C++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 <assert.h>
#include "SavAcct.h"

void open (SavingAccount*, int*, int);
void close (int*);

void debit(SavingAccount&);
void credit(SavingAccount&);
void monthly(SavingAccount&);
void annual(SavingAccount&);

double calAnlInt(SavingAccount&);
double calMonInt(SavingAccount&);

int getChoice();

void main() {
    SavingAccount *acctPtr;
    int choice, nbCust, ndx, nextCust=0;

    cout << setiosflags(ios::showpoint |
        ios::fixed) << setprecision(2);

    cout << "Enter number of Customers\n";
    cin >> nbCust;

    acctPtr = new SavingAccount [nbCust];
    assert(acctPtr !=0);
}

```

C++

```

choice = getChoice();
while(choice >=-2 && choice <=4) {
    if (choice < 0 ) {
        switch (choice) {
            case -2:open(acctPtr,&nextCust,nbCust);
                break;
            case -1:close(&nextCust);
                break;
        };
    }
    else {
        cout << "Enter Customer index\n" ;
        cin >> ndx;
        if (ndx < nextCust) {
            switch (choice) {
                case 0:debit(acctPtr[ndx]);
                    break;
                case 1:credit(acctPtr[ndx]);
                    break;
                case 2:monthly(acctPtr[ndx]);
                    break;
                case 3:annual(acctPtr[ndx]);
                    break;
            };
        }
        else
            cout << "Invalid Customer index\n" ;
    }
    choice = getChoice();
}
delete [] acctPtr;
}

```

C++

```

int getChoice() {
    int choice;
    cout << "\nEnter: \n"
        << "\t -2) Open an Account.\n"
        << "\t -1) Close an Account.\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 open(SavingAccount *aPtr,int *next,int max){
    unsigned nb;
    double bal, perc;

    if (*next < max) {
        cout << "Enter the Account Number: ";
        cin >> nb;
        cout << "Enter the Opening balance: ";
        cin >> bal;
        cout << "Enter the Interest Rate: ";
        cin >> perc;
        (aPtr + *next)->setAccount(nb, bal, perc);
        *next +=1;
    }
    else
        cout << "Cannot create account!!!\n";
}

```

C++

```
void close (int *next) {
    if ( *next > 0 )
        *next -=1;
}

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;
}
```

C++

```
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.balance *
           acctRef.annualInterestRate / 100.0;
}

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