ამოცანები პროგრამული პარადიგმებისთვის

ამოცანები ასემბლერზე

ამოცანა 1.(2011 წლის ფინალური) ასემბლერი(40 ქულა)
დაწერეთ ასემბლერის კოდი getText მეთოდისთვის. ცალკე გამოყავით თითოეული ხაზის შესაბამისი კოდი.

```
class block{
   block *build(int *a, char **c);
   string *getText(block b, short sh)
   {
      shorts[sh] += b.characters + (int *)shorts;
      return (string *) build(sh, b.characters)->shorts;
   }

   char **characters;
   short *shorts;
}
```

ამოცანა 6.(2011 წლის შუალედური) ორასემბლერი(25 ქულა)

დაწერეთ ასემბლერის კოდი WinslowHigh ფუნქციისთვის. ცალკე გამოყავით თითოეული ხაზის შესაბამისი კოდი.

```
struct teacher {
        short lauren[4];
        short *harper;
        char *harry[2];
};

static char *TrainedForBattle(struct teacher *lipschultz, short **history);
static void WinslowHigh(struct teacher buttle, struct teacher *marilyn)
{
        int k;
        if (k > 0) {
                marilyn->harper = buttle.lauren + 2;
        }
        buttle.harry[10] = TrainedForBattle(&buttle, &buttle.harper);
}
```

ამოცანა 13.(2012 წლის შუალედური) ასემბლერი(25 ქულა)

დაწერეთ ასემბლერის კოდი ქვემოთ მოცემული printAll ფუნქციისთვის. ცალკე გამოყავით თითოეული ხაზის შესაბამისი კოდი.

```
int printAll(void ** arr, int size) {
        int i = 0;
        for (; i<size; i++) {
                printf(*((char**)arr+i));
        }
        return *((int*)&i-1);
}
```

ამოცანა 19.(2012 წლის ფინალური) ასემბლერი(30 ქულა)

დაწერეთ ასემბლერის კოდი troll მეთოდისთვის. ცალკე გამოყავით თითოეული ხაზის შესაბამისი კოდი.

```
class WasNotExpected {
void * WasNotExpected::troll(WasNotExpected w, short sh) {
                sh = shorts[sh + w.count];
                troll(w, sh);

                return (char *)((&sh) + 2);
        }

        int count;
        short *shorts;
}
```

ამოცანა 25.(2012 წლის გადაბარება, 2013 წლის ფინალური) ასემბლერი(40 ქულა)
დაწერეთ ასემბლერის კოდი troll და getCount მეთოდებისთვის. ცალკე გამოყავით თითოეული ხაზის შესაბამისი კოდი.

```
class MyClass {
int MyClass::getCount() {
return this->count;
}

void * MyClass::troll(MyClass m, short sh) {
sh = shorts[sh + m.getCount()] + getCount();

return (char *)((&sh) + count);
        }

        int count;
        short *shorts;
}
```

ამოცანა 29.(2013 წლის შუალედური) პროლოგი(25 ქულა)

დაწერეთ ასემბლერის კოდი TrollMe ფუნქციისთვის. ცალკე გამოყავით თითოეული ხაზის შესაბამისი კოდი.

```
struct teacher {
        short lauren[4];
        short *harper;
        char *harry[2];
};

static void TrollMe(struct teacher buttle, struct teacher *marilyn)
{
        int k;
        if (k > 0) {
                marilyn->harper = buttle.lauren + 2;
        }
        buttle.harry[10] = malloc(&buttle, &buttle.harper);
}
```

ამოცანა 34.(2013 წლის გადაბარება,2015 წლის გადაბარება) ასემბლერი(40 ქულა)
დაწერეთ ასემბლერის კოდი something და everything მეთოდებისთვის. ცალკე გამოყავით თითოეული ხაზის შესაბამისი კოდი.

```
class HisClass {
short HisClass::something() {
return this->nothing + (short)this->thing;
}

void * HisClass::everything(HisClass c, int * hell) {
printf(hell + something(), c.nothing * c.thing);
return hell[c. nothing] + 17;
        }

        public:
                int thing;
                short nothing;
}
```

ამოცანა 41.(2014 წლის შუალედური) პროლოგი(30 ქულა)

დაწერეთ ასემბლერის კოდი TrollMe ფუნქციისთვის. ცალკე გამოყავით თითოეული ხაზის შესაბამისი კოდი.

```
struct teacher {
  short lauren[2];
  struct teacher *harper;
  char *harry[2];
};

static int TrollMe(struct teacher buttle[], int n)
{
  for(int i=0; i<n; i++){
    struct teacher * a = buttle + (int)buttle[i].harry;
    n = TrollMe((int)buttle, buttle->lauren);
  }
}
```

ამოცანა 45.(2014 წლის ფინალური) ასემბლერი(40 ქულა)
დაწერეთ ასემბლერის კოდი troll და getCount მეთოდებისთვის. ცალკე გამოყავით თითოეული ხაზის შესაბამისი კოდი.

```
class MyClass {
  int getCount() {
    return obj->count;
  }

  void * troll(short sh, MyClass * m) {
    MyClass s = obj[*(int *)troll((short)count, m->obj)];
    count = s.count + getCount();
    return obj;
  }
```

```
  int count;
  MyClass *obj;
};
```

ამოცანა 51.(2015 წლის შუალედური) პროლოგი(30 ქულა)
დაწერეთ ასემბლერის კოდი cola ფუნქციისთვის. ცალკე გამოყავით თითოეული ხაზის შესაბამისი კოდი.

```
typedef struct Komble{
  int kombal;
  struct Komble * next;
  char skynet[8];
}Komble;

int cola(short *a, Komble ko){
  int z = ko.skynet[ko.next->kombal];
  while(1){
    short x = *a + *(ko.skynet);
    z += cola(x, *(ko.next));
  }
  return z;
}
```

ამოცანა 55.(2015 წლის ფინალური) ასემბლერი(30 ქულა)
დაწერეთ ასემბლერის კოდი a და b მეთოდებისთვის. ცალკე გამოყავით თითოეული ხაზის შესაბამისი კოდი.

```
int printf(int x, ...);
class bla{
  int b(int z, bla m){
    int i = m.x[i];
    for (; i<x[i]; i++){
      temp[0]->temp[0]->a(i);
    }
    return m.temp[1]->x[2];
  }

  void a(int i){
    printf(i);
  }

  bla * temp[2];
  int x[3];
};
```

ამოცანები სქემაზე

ამოცანა 5.(2011 წლის ფინალური, 2014 წლის გადაბარება) ეძიებდე და ჰპოვებდე (40 ქულა)
თქვენი მიზანია გაარკვიოთ მოცემულ გრაფში ორ არსებობს თუ არა გზა ორ წვეროს შორის.
ამისათვის უნდა დაწეროთ ფუნქცია isway? რომელსაც სამი პარამეტრი გადაეცემა: გრაფი, საწყისი წვერო და საბოლოო წვერო. გრაფი მოცემული გაქვთ წვეროების წყვილების სიის სახით.
მაგალითი:
>(isway? ((1 2) (1 3) (2 4)) 1 4)
>#t

მითითება: BFS ან DFS

(define (isway? graph start finish)

ამოცანა 21.(2012 წლის ფინალური) სქემა (40 ქულა)
ა) დაწერეთ histogram ფუნქცია რომელსაც გადაეცემა ლისტი და რომელიც ევალუირდება ორი სიგრძის ლისტების ლისტში. დაბრუნებული ლისტის თითოეული ელემენტი უნდა იყოს ორ ელემენტიანი ლისტი, რომლის პირველი ელემენტია საწყისი ლისტის რომელიმე ელემენტი, ხოლო მეორე ელემენტი კი საწყის ლისტში იმ ელემენტის რაოდენობა. გაითვალისწინეთ რომ ევალუირებულ ლისტში ელემენტები არ უნდა მეორდებოდეს.
მაგალითი:
>(histogram ‘(1 2 3 4 1 2 3))
((1 2) (2 2) (3 2) (4 1)) - ორი ერთიანი, ორი ორიანი, ორი სამიანი და ერთი ოთხიანი
>(histogram ‘(2 1 2 2))
((2 3) (1 1)) - სამი ორიანი და ერთი ერთიანი.

არ არის აუცილებელი შემომავალი ლისტის ელემენტები პრიმიტიული ტიპის იყოს:
>(histogram ‘(1 (1 2) 1 (3)))
((1 2) ((1 2) 1) ((3) 1))) - ორი ერთიანი, ერთი (1 2) და ერთი (3).

(define (histogram list)

ბ) თქვენი მიზანია დაწეროთ multiply ფუნქცია რომელსაც გადაეცემა ინტეჯერი n და ლისტი. ფუნქცია უნდა ევალუირდებოდეს ახალ ლისტში რომელიც მიიღება საწყისი ლისტიდან თითოეულ ელემენტს თუ შევცვლით n ცალი მსგავსი ელემენტით.
მაგალითად:
>(multiply 2 ‘(a 3 5))
(a a 3 3 5 5)
>(multiply 3 ‘((1) 3))
((1) (1) (1) 3 3 3)

(define (multiply n list)

ამოცანა 26.(2012 წლის გადაბარება, 2013 წლის ფინალური) სქემა (40 ქულა)
ა) დაწერეთ union ფუნქცია რომელსაც გადაეცემა რამდენიმე სიმრავლე(ლისტი) და ევალუირდება სიმრავლეში(ლისტში) რომელიც არის გადაცემული სიმრავლეების გაერთიანება.
მაგალითად:
(union) ⇒ ()
(union '(a b c)) ⇒ (a b c)
(union '(2 5 4) '(9 4 3)) ⇒ (2 5 9 4 3)
(union '(1 2) '(2 4) '(4 8)) ⇒ (1 2 4 8)

ბ) თქვენი მიზანია დაწეროთ intersect ფუნქცია რომელიც გამოითვლის ორი სიმრავლის თანაკვეთას. ფუნქციას გადაეცემა ორი ლისტი და ევალუირდება ლისტში.
მაგალითი:

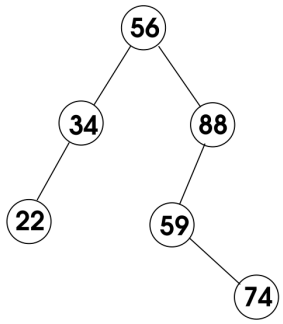
(intersect '(a b c) '(a b d e)) ⇒ (a b)

(intersect '((a b) c (d)) '(a b c e (d))) ⇒ (e (d))

ამოცანა 5.(2013 წლის გადაბარება) მიმატება (40 ქულა)
თქვენი ამოცანაა დაწეროთ addition ფუნქცია რომელსაც შეუძლია დიდი რიცხვების შეკრება. ჩათვალეთ რომ თითოეული რიცხვი წარმოდგება ციფრების ლისტის სახით. მაგალიღად რიცხვი 234 -> ‘(2 3 4), 17 -> ‘(1 7)… ასევე ჩათვალეთ რომ სქემაში ორ მილიარდზე მეტი რიცხვების შეკრების დროს ხდება მეხსიერების გადავსება (overflow) რაც ნიშნავს იმას რომ მაგალითად (* 1000000 1000000) შეცდომას ამოაგდებს. addition ფუნქციას შეიძლება გადაეცეს რამოდენიმე ლისტი. ის უნდა ევალუირდეს ამ ლისტების შესაბამისი რიცხვების ჯამის შესაბამის ლისტში. მაგალითად:
(addition ‘(6 3) ‘(5 8)) => ‘(1 2 1)
(addition ‘(3) ‘(1 7 7)) => ‘(1 8 0)
(addition ‘(4) ‘(3 3) ‘(1 2 3) ‘(4 3 2 1)) => ‘(4 4 8 1)

ამოცანა 46.(2014 წლის ფინალური) სქემა (40 ქულა)

თქვენი მიზანია generic ორობითი ძებნის ხის იმპლემენტაცია. დასაწერი გაქვთ ორი ფუნქცია ძებნის და ელემენტის ჩამატების.
ორობითი ძებნის ხე შეიძლება წარმოვადგინოთ 3 ელემენტის მქონე ლისტის სახით. პირველი ელემენტი იყოს რუთის მარცხენა ქვე-ხე მეორე ელემენტი იყოს რუთის მნიშვნელობა მესამე ელემენტი იყოს მარჯვენა ქვე-ხე. ალბათ გასაგებია რომ პირველი და მესამე ელემენტები თავის მხრივ 3 ელემენტიანი ლისტებია. ხის ფოთლებიც 3 ელემენტიანი ლისტით წარმოდება - პირველი და მესამე ელემენტები ცარიელი ლისტებია ხოლო მეორე ელემენტი კი მნიშვნლეობა. მაგალითად მოცემული ხე სქემაში შეიძლება წარმოვადგინოთ შემდეგნაირად:



(((() 22 ()) 34 ()) 56 ((() 59 (() 74 ())) 88 ()))

ზედმეტი სფეისები სტრუქტურის უკეთ აღსაქმელად არის დაწერილი.

ა) თქვენი ამოცანაა დაწეროთ ფუნქცია რომელსაც გადაეცემა ორობითი ძებნის ხე და ელემენტი და რომელიც ევალუირდება ორობით ხეში რომელშიც ეს ელემენტი ჩამატებულია.

ბ) თქვენი ამოცანაა დაწეროთ ფუნქცია რომელსაც გადაეცემა ორობითი ძებნის ხე და ელემენტი და რომელიც აბრუნებს #t თუკი ეს ელემენტი მოიძებნა ორობით ხეში ხოლო #f თუკი არ მოიძებნა

ჩათვალეთ რომ შემომავალი ორობითი ხეები ბალანსირებულია. დაწერეთ ფუნქციები რომლების მუშაობის სირთულეა O(logN) სადაც N ხეში არსებული ელემენტების რაოდენობაა.

ამოცანა 56.(2015 წლის ფინალური) სქემა (40 ქულა)
თქვენი ამოცანაა კომივოიაჟერს დაეხმაროთ ქალაქებს შორის გზის გაგნებაში. მოცემული გაქვთ ქალაქების გრაფი - მანძილებით ქალაქებს შორის, რომლებიც მთელი რიცხვებია. ჩათვალეთ რომ ქალაქები გადანომრილია 1-დან n-მდე, სადაც n ქალაქების რაოდენობაა. თქვენ უნდა იპოვოთ უმოკლესი გზა რომელიც პირველი ქალაქიდან იწყება და ყველა ქალაქს გაივლის ზუსტად ერთხელ.
ამისათვის პროგრამირების ენა სქემაში უნდა დაწეროთ salesman ფუნქცია რომელსაც გადაეცემა ქალაქების რაოდენობა n და გრაფის სტრუქტურა და რომელიც ევალურიდება ლისტში. salesman ფუნქციის მიერ ევალუირებული ლისტი უნდა იყოს ქალაქების ნომრების თანმიმდევრული ჩამონათვალი - ანუ გზა რომელსაც კომივოიაჟერი გაივლის. ეს ლისტი უნდა იწყებოდეს 1-იანით, რადგან კომივოიაჟერი

პირველი ქალაქიდან იწყებს მოძრაობას და უნდა შედგებოდეს n წევრისგან, რადგან კომივოიაჟერმა ყველა ქალაქი უნდა გაიაროს. შეგიძლიათ ჩათვალოთ რომ ასეთი გზა აუცილებლად არსებობს.
ჩათვალეთ რომ გრაფი მიმართულია და ის მოცემულია უბრალოდ წიბოების ლისტით. თოთოეული წიბო ცხადია ორ მეზობელ ქალაქს შორის გზას ასახავს. შესაბამისად თითოეული წიბო თავის მხრივ არის სამი სიგრძის ლისტი, რომლის პირველი ელემენტიც არის გზის საწყისი ქალაქი ნომერი, მეორე ელემენტი გზის საბოლოო ქალაქის ნომერი, ხოლო მესამე ელემენტი კი გზის სიგრძე. მაგალითად წიბო (2 3 4) ნიშნავს რომ არსებობს გზა მეორე ქალაქიდან მესამე ქალაქში რომლის სიგრძეა 4.

მაგალითი:
(salesman 3 '((1 2 1) (2 3 2) (1 3 1) (3 2 1))) = > (1 3 2)

ამოცანა 59.(2015 წლის გადაბარება) ყველა ჯამი (40 ქულა)
გვინდა რიცხვი დავშალოთ ნატურალური რიცხვების ჯამის სახით, ოღონდ ყველა შესაძლო ხერხით. მაგალითად რიცხვი 3 შეიძლება დაიშალოს სამი შესაძლო ხერხით: 1+1+1, 1+2, 2+1; ხოლო რიცხვი 4 კი შვიდი შესაძლო ხერხით: 1+1+1+1, 1+1+2, 1+2+1, 2+1+1, 1+3, 3+1, 2+2.
თქვენი ამოცანაა დაწეროთ ფუნქცია allsum რომელსაც გადაეცემა რიცხვი და რომელიც აბრუნებს ყველა შესაძლო “დაშლას" ლისტის სახით. გაითვალისწინეთ რომ დაბრუნებულ ლისტში თითოეული “დაშლა” თავის მხრივ ნატურალური რიცხვებისგან შემდგარი ლისტია სადაც მიმდევრობას მნიშვნელობა აქვს.
(allsum 3) -> ((1 1 1) (1 2) (2 1))
(allsum 4) -> ((1 1 1 1) (1 1 2) (1 2 1) (2 1 1) (1 3) (3 1) (2 2))

ასევე გაითვალისწინეთ ის ფაქტი, რომ პროგრამა უნდა დაიწეროს ენაზე scheme და არ უნდა დაირღვეს ფუნქციონალური პარადიგმა.

ამოცანა 60(არსად არ ყოფილა). ჯამები (50 ქულა)

თქვენი ამოცანაა მოცემული რიცხვი დაშალოთ ნატურალური რიცხვების ჯამის სახით, ოღონდ ყველა შესაძლო ხერხით. ამისათვის გადმოგეცემათ ნატურალური რიცხვების სია, რომლებიც შეგიძლიათ გამოიყენოთ დაშლისთვის. სიაში არსებული ელემენტები შეგიძლიათ მრავალჯერ გამოიყენოთ. მაგალითად თუკი გვინდა რიცხვი 3-ის დაშლა ნატურალური რიცხვებით 1 და 2 ეს შეგვიძლია გავაკეთოთ 2 გზით:

3 = 1 + 1 + 1

3 = 1 + 2

ყურადღება მიაქციეთ იმ ფაქტს, რომ დაშლა 3 = 1 + 2 და დაშლა 3 = 2 + 1 ერთიდაიგივეა. ანუ დაშლაში მიმდევრობას მნიშვნელობა არ აქვს.

თუკი 4-ის დაშლა გვინდა რიცხვებით 1,2 და 3 ეს შეიძლება 4 გზით

4 = 1 + 1 + 1 + 1

4 = 1 + 1 + 2

4 = 1 + 3

4 = 2 + 2

დაწერეთ all-sum-from-list ფუნქცია რომელსაც გადაეცემა რიცხვი რომლის დაშლაც გვინდა და ნატურალური რიცხვების ლისტი რითიც გვინდა რომ დავშალოთ. ამ ფუნქციამ უნდა დაბრუნოს ყველა “დაშლა” ლისტის სახით. გაითვალისწინეთ ის, რომ თითოეული “დაშლა” თავის მხრივ ნატურალური რიცხვების ლისტია სადაც რიცხვების მიმდევრობას მნიშვნელობა არ აქვს. არ არის აუცილებელი ფუნქციამ დააბრუნოს მხოლოდ განსხვავებული დაშლები მთავარია, რომ ფუნქციის მიერ დაბურნებული ყველა დაშლა ვალიდური იყოს(ანუ ჯამი იყოს გადმოცემული რიცხვი და დაშლაში შემავალი ელემენტები იყვნენ გადმოცემული ლისტიდან) და არც ერთი დაშლა არ იყოს გამოტოვებული.

შეზღუდვები:
პროგრამა უნდა დაიწეროს ენაზე scheme და არ უნდა დაირღვეს ფუნქციონალური პარადიგმის პრინციპები.
ჩათვალეთ, რომ გადაცემულ ლისტში ელემენტების რაოდენობა ნაკლებია 21-ზე. ასევე ჩათვალეთ, რომ “დასაშლელი” რიცხვის განაყოფი ლისტში არსებულ მინიმალურ ელემენტთან ნაკლებია 21-ზე.
ამ შეზღუდვების გათვალისწინებით თქვენი ამოხსნა თითოეულ ტესტს უნდა ასრულებდეს არაუმეტეს 1 წამში.
თქვენი კოდი დაწერეთ all-sum-from-list.scm ფაილში. შეგიძლიათ გამოიყენოთ ნებისმიერი ედიტორი. კომპილაციისთვის გამოიყენეთ კავას ინტერპრეტატორი, რომელიც თქვენს დესკტოპზეა და შეგიძლიათ გაუშვათ kawa.jar ფაილზე უბრალოდ მაუსის დაკლიკებით.

ტესტირებისთვის გამოიყენეთ შემდეგი ტესტები:
(all-sum-from-list 3 ‘(1 2)) -> ((1 1 1) (1 2)) --- სწორი იქნება შემდეგი პასუხიც ((1 1 1) (1 2) (2 1))
(all-sum-from-list 4 ‘(1 2 3)) - > ((1 1 1 1) (1 1 2) (1 3) (2 2))
(all-sum-from-list 5 ‘(2 3)) -> ((2 3))
(all-sum-from-list 6 ‘(4 5)) -> ()

ამოცანა 2. (2011 წლის ფინალური, 2013 წლის შუალედური) პრიორიტეტული რიგი(40 ქულა)
ამ ამოცანაში თქვენი მიზანია პრიორიტეტული რიგის იმპლემენტაცია.
პრიორიტეტული რიგის რეალიზაცია სულ 4 ფუნქციას საჭიროებს ესენია - სტრუქტურის შექმნის, წაშლის, სტრუქტურიდან ელემენტის ამოღების და ჩამატების ფუნქციები.
გაითვალისწინეთ, რომ პრიორიტეტულ რიგში, ჩვეულებრივი რიგისგან გასხვავებით, თითოეულ ელემენტს გააჩნია საკუთარი პრიორიტეტი და ელემენტების ამოღება ხდება პრიორიტეტის მიხედვით და არა იმის მიხედვით თუ რომელი ელემენტი როდის დაემატა სტრუქტურას. შესაბამისად ყოველი ელემენტის დამატებისას აუცლებელია მითითებულ იქნას ელემენტის პრიორიტეტიც(მთელი დადებითი რიცხვი). ელემენტის ამოღების მეთოდმა კი უნდა დააბრუნოს სტრუქტურაში არსებული ელემენტებიდან ყველაზე დიდი პრიორიტეტის მქონე ელემენტი.
თქვენ მოცემული გაქვთ ფუნქციების ჩამონათვალი რომელის იმპლემენტაციაც უნდა დაწეროთ:

```
typedef struct {

} pqueue;
void PQueueNew              - პრიორიტეტული რიგის ინიციალიზაცია.
void PqueueEnqueue          - ახალი ელემემენტის დამატების ფუნცია
void PqueueDequeue          - ელემენტის ამოღების ფუნქცია
void PqueueDispose          - მეხსიერების გათავისუფლება
```

ყურადღება მიაქციეთ იმ ფაქტს რომ ფუნქციების ზუსტი სიგნატურები არ არის მოცემული შესაბამისად მათი მოფიქრება თქვენი ამოცანაა. აგრეთვე ყურადღება მიაქციეთ იმ ფაქტს რომ კოდი უნდა დაიწეროს C-ში და არ გაქვთ არანაირი არასტანდარტული სტრუქტურების გამოყენების უფლება(იგულისხმება vector, hashset,...).

ამოცანა 3.(2011 წლის ფინალური, 2013 წლის შუალედური) გძელი სიტყვა მოკლედ ითქმის (30 ქულა)
პრიორიტეტული რიგის გამოყენებით მოახდინეთ შემდეგი ფუნქციის იმპლემენტაცია:
char *getLongestWord(char ** words, int wordsNum);
ფუნქციას getLongestWord გადაეცემა ორი არგუმენტი char*-იანების მასივი(words) და ამ მასივის სიგრძე(wordsNum) და აბრუნებს words მასივში ყველაზე გრძელ სიტყვას. ამ ფუნქციის იმპლემენტაციისთვის აუცილებელია გამოიყენოთ პრიორიტეტული რიგი. მასივის თითოეული სიტყვა დაამატეთ რიგში საკუთარი სიგრძის მქონე პრიორიტეტით და შემდეგ დააბრუნეთ ყველაზე პრიორიტეტული(ანუ ყველაზე გრძელი) სიტყვა.

```
char *getLongestWord(char ** words, int wordsNum){
```

ამოცანა 4.(2011 წლის ფინალური) თამაში(30 ქულა)
მოცემული გაქვთ *hero.h, hero.c* და *game.c* ფაილები:

```
/**********************************hero.h*************************************/
typedef struct {
        int id;
        double currLife;
} hero;
void Hero(hero *c, int id, int initLife);

/**********************************hero.c*************************************/
#include "hero.h"
void HeroNew(hero *c, int id, int initLife) {
        c->id = id;
        c->currLife = initLife;
}

/**********************************game.c*************************************/
#include "vector.h"
typedef struct {
        vector heros;
} game;
void HeroAttack(game* g, hero *attacker, hero *victim) {
//მსხვერპლის სიცხოცხლეს ამცირებს, შემტევის სიცოცხლის 10% ით
        victim->currLife -= attacker->currLife*10/100;
//თუ მსხვერპლის სიცოცხლე <= 0, იგი უნდა წაიშალოს თამაშიდან
        if (victim->currLife <= 0) {
```

```
                VectorDelete(&g->heros, victim);
        }
}
```

hero - სტრუქტურა აღწერს მოთამაშეს რომელსაც აქვს უნიკალური id და საწყისი სიცოცხლე.
game - სტრუქტურა აღწერს თამაშში მონაწილე გმირების სიას.

თამაშის პროცესში გმირები ახდენენ ერთმანეთზე თავდასხმას, რის შედეგადაც ზოგიერთი მათგანი იღუპება.

არსებული კოდი სწორად იმუშავებს მხოლოდ ერთ ნაკადიანი პროგრამისთის, თუ HeroAttack ფუნქციის გამოძახება მოხდება პარალელურად რამოდენიმე ნაკადიდან, პროგრამა აღარ იქნება გამართული. თქვენი ამოცანაა არსებული კოდი შეცვალოთ ისე, რომ HeroAttack ფუნქციამ გამართულად იმუშაოს ბევრ ნაკადიანი პროგრამის დროსაც.

ამოცანა 7.(2011 წლის შუალედური) მაქსიამალური სიგრძის მასივი(20 ქულა)
თქვენი მიზანია დაწეროთ პროგრამა რომელიც დაბეჭდავს რა მაქსიმალური ზომის int ტიპის მასივის შექმნა არის შესაძლებელი ჰიპში, პროგრამის გაშვების მომენტში. ამ დავალების შესასრულებლად შეგიძლიათ გამოიყენოთ malloc ფუნქცია.
ჩათვალეთ, რომ თქვენი პროგრამის გაშვების შემდეგ არც ერთი სხვა პროგრამა ჰიპს არ ეხება.
ასევე ჩათვალეთ რომ თქვენი მასივის მაქსიმალური წონა შეიძლება იყოს 4GB.
მოიფიქრეთ ისეთი ალგორითმი რომელიც 1 წუთზე ნაკლებ დროში შეასრულებს სამუშაოს.

ამოცანა 8.(2011 წლის შუალედური) ხაშმაპი(30 ქულა)
თქვენი მიზანია ჰეშმეფის რეალიზაციის დაწერა. ალბათ კარგად იცით ეს სტრუქტურა რასაც წარმოადგენს. ამისათვის თქვენ შეგიძლიათ გამოიყენოთ მესამე დავალებაში უკვე დაწერილი ჰეშსეტ სტრუქტურა და მისი ფუნქციები.
თქვენ უნდა დაწეროთ 3 მეთოდი: ახალი ჰეშმეფის გაკეთება, ჰეშმეფის წაშლა, ჰეშმეფში ჩამატება. ყურადღება მიაქციეთ იმ ფაქტს რომ ჰეშმეფი შეიცავს არა ცალკეულ ელემენტს, არამედ key/value წყვილს და მასში ვერ იქნება ორი წყვილი რომელთაც ერთი და იგივე key აქვთ. ამიტომ ჰეშსეტი ცოტა ჭკვიანურად უნდა გამოიყენოთ.
თვითონ ჰეშმეფ სტრუქტურის მოფიქრება თქვენთვის მომინდვია.

```
typedef int (*HashMapHashFunction)(const void *elem, int numBuckets);
typedef int (*HashMapKeyCompareFunction)(const void *elem1, const void* elem2);
```

```
typedef void (*HashMapKeyFreeFunction)(void *elem);
typedef void (*HashMapValueFreeFunction)(void *elem);


void HashMapNew(hashMap *hm, int elemSize, int numBuckets,
                                        HashMapHashFunction hash,
                                        HashMapKeyCompareFunction keyCompare,
                                        HashMapKeyFreeFunction keyFree,
                                        HashMapValueFreeFunction valueFree);
void HashMapDispose(hashMap *hm);
void HashMapAdd(hashMap *hm, const void *key, const void *value);
typedef struct{



}hashMap;
```

ამოცანა 9.(2011 წლის შუალედური) ციკლუ რისია(25 ქულა)
თქვენი ამოცანაა ციკლური ლინკდ-ლისტი(ბმული სია) იპოვოთ მეხსიერების გარკვეულ ბლოკში. ჩათვალეთ რომ მეხსიერების ბლოკი მოცემულია ორი გლობალური ცვლადის საშუალებით void *blockStart და void *blockEnd. ლინკდ-ლისტი კი მოცემულია შემდეგი სტრუქტურის სახით:

```
typedef struct{
        char *name;
        char grade;
        node *next;
}node;
```

იმის გაგება შესაძლებელია თუ არა მიმთითებლის დერეფერენსი არც თუ ისე მარტივია. სიმარტივისთვის ჩავთვალოთ რომ ასეთი ფუნქცია უკვე დაწერილია, რომელიც აბრუნებს 1-ს იმ შემთხვევაში თუ დერეფერენსი შესაძლებელია და 0-ს წინააღმდეგ შემთხვევაში.

```
int isDereferencable(void *ptr);
```

ყურადღება მიაქციეთ იმ ფაქტს რომ თქვენ უნდა მოძებნოთ ნებისმიერი სიგრძის ციკლური ლინკდ-ლისტი რომლის თითოეული წევრიც ჩვენს მიერ აღწერილი ტიპისაა.

ამოცანა 10.(2011 წლის შუალედური) მოკლეს(20 ქულა)
ა) (2013 წლის ფინალურიც)
რას დაბეჭდავს შემდეგი პროგრამა? თუ იგი ეშვება bigendian სისტემაზე

```
int main()
{
        int a[10];
        a[0] = 3;
        *((char*)a + 1) = 12;

        printf("%d", strlen((char*)a));

        return 0;
}
```

ბ) ახსენით რატომ არის არასწორი შემდეგი ჩანაწერი:

```
void * a = malloc(12);
a++;
```

ამოცანა 11.(2012 წლის შუალედური) უცებ გასაკეთებელი(15 ქულა)
ა) რას დაბეჭდავს შემდეგი პროგრამა?

```
int main() {
	char * text = "abcdefgh";
	int m = 1;
	for (int i=0; i<9; i++)
		m = m**(text+i);
	cout << m << endl;
	return 0;
}
```

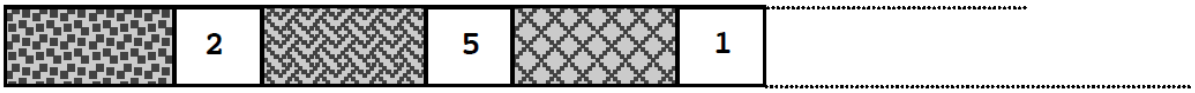
ბ) რას დაბეჭდავს შემდეგი პროგრამა?

```
int main() {
	char * text = "abcdefgh";
	text = text + (*(text + 3) - *text);
	cout << text << endl;
	return 0;
}
```

ამოცანა 12.(2012 წლის შუალედური) სტრინგების მანიპულაცია(20 ქულა)
თქვენი ამოცანაა პროგრამირების ენა C-ში დაწეროთ Modife(char * word, int n) ფუნქცია. ფუნქციამ უნდა მოაჭრას word სტრინგს n სიგრძის პრეფიქსი და სუფიქსი იმ შემთხვევაში თუკი ეს პრეფიქსი და სუფიქსი ერთმანეთის ტოლია.
ანუ თუკი მოცემული სტრიგნგის საწყისი და ბოლო n სიმბოლო ერთიდაიგივეა მაშინ word სტრინგი უნდა გარდაიქმნეს ტრინგად რომელსაც ეს პრეფიქსი და სუფიქსი არ ექნება.
მაგალითად: n=2, სტრინგი “abcab” უნდა გარდაიქმნას სტრინგად “c”.

```
void Modife(char *word, int n){
```

ამოცანა 14.(2012 წლის შუალედური) კალათები(40 ქულა)
თქვენი ამოცანაა შექმნათ Bucket სტრუქტურა. ეს არის სტრუქტურა რომლის საშუალებითაც შეიძლება გენერიკ ობიექტების შენახვა. სტრუქტურა ინახავს მხოლოდ უნიკალურ ობიექტებს და თითოეული ობიექტის რაოდენობას. მოახდინეთ სტრუქტურის ისეთი იმპლემენტაცია, რომლისთვისაც მეხსიერება შემდეგნაირად გამოიტყურება :
ელემენტი1(? ბაიტი), ელემენტი1-ის რაოდენობა(4 ბაიტი); ელემენტი2(? ბაიტი), ელემენტი2-ის რაოდენობა(4 ბაიტი)....



თქვენი მიზანია მოიფიქროთ თუ როგორია ამ სტრუქტურის ველები და გააკეთოთ შემდეგი ფუნქციების იმპლემენტაცია:
void BucketNew(Bucket *b, ცვლადები); - ახალი სტრუქტურის შექმნა/ინიციალიზირება
void BucketAdd(Bucket *b, void *elem); - ელემენტის დამატება, რომლის დროსაც ვამოწმებთ არის თუ არა მსგავსი ელემენტი სტუქტურაში. თუკი არის მაშინ ელემენტების რაოდენობას ვზრდით ერთით, წინააღმდეგ შემთხვევაში კი ვამატებთ ახალ ელემენტს რაოდენობით 1.
void* BucketMax(Bucket *b); - აბრუნებს ელემენტს რომელიც ყველაზე დიდი რაოდენობით გვაქვს სტრუქტურაში.
void BucketDispose(Bucket *b); - სტრუქტურის მიერ დაკავებული მეხსიერების განთავისუფლება.

ცხადია რომ იმპლემენტაცია უნდა მოხდეს C-ზე და იმპლემენტაციის დროს არ გაქვთ უფლება გამოიყენოთ ისეთი სტრუქტურები როგორიცაა Vector,Set,Map...

typedef struct{

}Bucket;

ამოცანა 15.(2012 წლის შუალედური) ყველაზე ხშირი გვარი(30 ქულა)
მოცემულია ადამიანების სია, თქვენი მიზანია Bucket სტრუქტურის საშუალებით იპოვოთ ერთერთი ადამიანის სახელი და გვარი რომელსაც ყველაზე გავრცელებული გვარი აქვს.

```
typedef struct{
        char *name;
        char *surname;
}Person;
```

უნდა გააკეთოთ getPerson ფუნქციის იმპლემენტაცია რომელსაც გადაეცემა Person სტრუქტურების მასივი და ამ სტრუქტურების რაოდენობა და რომელიც აბრუნებს ყველაზე ხშირი გვარის მქონე ერთერთი ადამიანის პრაბელით გამოყოფილ სახელს და გვარს.

```
char * getPerson(struct Person people[], int n){
```

ამოცანა 16.(2012 წლის ფინალური, 2014 წლის გადაბარება) C-სტრინგი(30 ქულა)
თქვენი მიზანია დაწეროთ replace ფუნქცია რომელსაც გადაეცემა სამი C-სტრინგი და რომელიც აბრუნებს C-სტრინგს. ფუნქცია პირველს სტრინგში ეძებს ყოველ შემხვედრ მეორე სტრინგს და ცვლის მას მესამე სტრინგით.
მაგალითად გამოძახებისას replace(“ababa”, “b”, “cc”) ფუნქციამ უნდა დააბრუნოს “accacca”

```
char* replace(char* st, char* find, char* repl){
```

ამოცანა 17.(2012 წლის ფინალური) რას დაბეჭდავს? (20 ქულა)

რას დაბეჭდავს შემდეგი პროგრამები?
გაითვალისწინეთ რომ არქიტექტურა შეიძლება იყოს როგორც 32 ბიტიანი ასევე 64 ბიტიანი. თუკი პროგრამის კომპილაციის ან მსვლელობის დროს მოხდება შეცდომა დაწერეთ თუ რა შეცდომაა და აღწერეთ თუ რატომ ხდება.
ჩათვალეთ რომ კომპიუტერის მეხსიერება(RAM) არის 1Gb.

ა)
```
int main(){
        int size = 4;
        void * a = malloc(2);
        while(size<1*1024*1024*1024){
        realloc(a, size);
        if (a==NULL) break;
        size = size * 2;
}
printf(“%d”,size);
}
```

ბ)
```
int main() {
int* p;
p = (int*)malloc(sizeof(int));
*p = 10;
printf("%d", *p);
}
```

ამოცანა 18.(2012 წლის ფინალური, 2014 წლის გადაბარება) სიმრავლე პლიუსი(40 ქულა)
ამ ამოცანაში თქვენი მიზანია სიმრავლე პლიუსის იმპლემენტაცია. სიმრავლე პლიუსის იმპლემენტაცია სულ 4 ფუნქციის რეალიზაციას საჭიროებს, ესენია - სტრუქტურის შექმნის, წაშლის, სტრუქტურაში ელემენტის ჩამატების მეთოდები და მეთოდი რომლის საშუალებითაც შეგვიძლია გავიგოთ შეიცავს თუ არა სტრუქტურა რომელიმე ელემენტს.
ჩვეულებრივი სიმრავლისგან განსხვავებით სიმრავლე პლიუსში შესაძლებელია ნებისმიერი ტიპის ობიექტის დამატება. მაგალითად სტრუქტურაში შესაძლებელია ერთდროულად შევინახოთ როგორც int ტიპის ცვლადები ასევე სტრინგები და სხვადასხვა სტანდარტული თუ არასტანდარტული ტიპის ცვლადები.

თქვენ მოცემული გაქვთ ფუნქციების ჩამონათვალი რომელის იმპლემენტაციაც უნდა დაწეროთ:

```
typedef struct {

} setp;
```

void SetPNew - სიმრავლე პლიუსის ინიციალიზაცია.
void SetPAdd - ახალი ელემემენტის დამატების ფუნცია
bool SetPContains - შემოწმება არის თუ არა ელემენტი სიმრავლეში
void SetPDispose - მეხსიერების გათავისუფლება

ყურადღება მიაქციეთ იმ ფაქტს რომ ფუნქციების ზუსტი სიგნატურები არ არის მოცემული შესაბამისად მათი მოფიქრება თქვენი ამოცანაა. გასათვალისწინებელია რომ ელემენტის დამატების დროს მხოლოდ ამ ელემენტზე მიმთითებლის გადაცემა საკმარისი არ არის, რადგან სიმრავლე ვერ მიხვდება ელემენტის ზომას და შესაბამისად მის კოპიას ვერ შეინახავს.
აგრეთვე ყურადღება მიაქციეთ იმ ფაქტს რომ კოდი უნდა დაიწეროს C-ში და არ გაქვთ არანაირი არასტანდარტული სტრუქტურების გამოყენების უფლება(იგულისხმება vector, hashset,...).

ამოცანა 20.(2012 წლის ფინალური) თამაში(40 ქულა)
თქვენი მიზანია გააუმჯობესოთ ბრაუზერის მუშაობა. მოცემული გაქვთ makeHttpRequest და processAllHttpsRequeusts ფუნქციები, რომელთა გამოყენება ხდება შემდეგი ინსტრუქციების მიხედვით:
M ცალი ნაკადიდან მუდმივად ხდება makeHttpRequest ფუნქციის გამოძახება. makeHttpRequest ფუნქცია urlData რიგში ამატებს გადმოცემულ სტრუქტურას, იბლოკება მანამ სანამ processAllHttpsRequeusts ფუნქცია არ ჩამოტვირთავს შესაბამის მონაცემებს და შემდეგ იძახებს displayData მეთოდს.
processAllHttpsRequeusts ფუნქციის გამოძახება მუდმივად ხდება მხოლოდ ერთი ნაკადიდან, იგი იბლოკება სანამ რიგში დამატებული ელემენტების რაოდენობა N-ზე ნაკლებია. როგორც კი რაოდენობა გახდება N ის ტოლი, ან მეტი, ფუნქცია რიგრიგობით იწყებს მონაცემების ჩამოტვირთვას სანამ რიგი არ გაცარიელდება.
თქვენი ამოცანაა დაასრულოთ makeHttpRequest და processAllHttpsRequeusts ფუნქციების რეალიზაცია, ისე რომ სისტემამ გამართულად იმუშაოს

```
struct data {
        char * url;
        void * data;
        int dataSize;


}
/* global shared data goes here */
queue urlData;  // supports size(), push(), pop(), and any other queue
// operations you need



makeHttpRequest(data *d) {


        urlData.push(d);
```

```
displayData(d);  //უზრუნველყოფს d->data ში წაკითხული მონაცემების
//ვიზუალიზაციას


}

processAllHttpsRequeusts() {

        /* service all the requests in the queue */
        while (urlData.size() != 0) {


                data* d = urlData.pop();


                readHttpRequest(d);      //უზრუნველყოფს d->url დან მონაცემების
// წაკითხვას და d->data ში ჩაწერას


        }


}
```

ამოცანა 22.(2012 წლის გადაბარება, 2013 წლის ფინალური) ექსპერიმენტი (40 ქულა)
ა) თქვენი მიზანია დაწეროთ shuffle ფუნქცია რომელსაც გადაეცემა გენერიკ ტიპის მასივი და რომელიც ამ მასივში რანდომით აურევს ელემენტებს. მასივის სიგრძეა n. თითოეული i-ური ელემენტისთვის დააგენერირეთ რანდომ რიცხვი j 1-დან n-მდე და გაუცვლეთ ადგილი i-ურ და j-ურ ელემენტებს. ამისთვის თქვენ შეგიძლიათ გამოიყენოთ RandomInteger(a,b) ფუნქცია რომელიც აბრუნებს რანდომ მთელ რიცხვს a-დან b-ს ჩათვლით.

```
void shuffle(void * base, int n, int elemSize){
```

ბ) როგორც ამბობენ წინადადების წაკითხვა მაინც არის შესაძლებელი თუკი მსაში შმეალავი სტიყბევს აუვერვთ აოსბეს. თქვენი მიზანია shuffle ფუნქციის გამოყენებით დაწეროთ jumbleSentence ფუნქცია რომელსაც გადაეცემა წინადადება და რომელიც ამ წინადადების თითოეულ სიტყვას ასოებს აურევს. ჩათვალეთ რომ სიტყვები თითო პრაბელით არის გამოყოფილი.

```
void jumbleSentence(char * sentence){
```

ამოცანა 23.(2012 წლის გადაბარება) რას დაბეჭდავს? (20 ქულა)

რას დაბეჭდავენ შემდეგი C++ პროგრამები? თუკი პროგრამის კომპილაციის ან მსვლელობის დროს მოხდება შეცდომა დაწერეთ თუ რა შეცდომაა და აღწერეთ თუ რატომ ხდება.

ა) (2013 წლის შუალედურიც)

```
#include<iostream>
using namespace std;
void swap(char * &str1, char * &str2){
  char *temp = str1;
  str1 = str2;
  str2 = temp;
}
int main(){
  char *str1 = "butkashio";
  char *str2 = "imnairio";
  swap(str1, str2);
  cout<<"shedi qalo "<<str1<<endl;
  cout<<"ara var "<<str2<<endl;
  return 0;
}
```

პასუხი:

ბ)

```
#include<iostream>
using namespace std;
class Test{
private:
  int x;
  int y;
public:
  Test(int x = 0, int y = 0) { this->x = x; this->y = y; }
  Test setX(int a) { x = a; return *this; }
  Test setY(int b) { y = b; return *this; }
  void print() { cout<< "x = " <<x<< " y = " <<y<< endl; }
};
int main()
{
  Test obj1;
  obj1.setX(10).setY(20);
  obj1.print();
  return 0;
}
```

პასუხი:

ამოცანა 24.(2012 წლის გადაბარება, 2013 წლის ფინალური) უსაფრთხო ვექტორი(40 ქულა)

მოცემული გაქვთ, vector სტრუქტურის მეთოდები. თქვენი მიზანია თითოეული მეთოდი (VectorNth, VectorReplace, VectorInsert, VectorDelete) შეცვალოთ ისე, რომ მიღებული რეალიზაცია გახდეს “threadsafe” და არ დაკარგოს ეფექტურობა. ის უნდა აკმაყოფილებდეს შემდეგ პირობებს:

ჩამოთვლილი მეთოდები გამართულად უნდა მუშაობდეს იმ შემთხვევაშიც, როცა vector სტუქტურაზე მიმართვა ხდება ერთდროულად რამოდენიმე ნაკადიდან.

ერთდროული შესრულების შემთხვევაში VectorNth და VectorReplace მეთოდები არ უნდა აფერხებდნენ ერთმანეთს, თუ მიმართვა არ ხდება ერთსა და იმავე პოზიციაზე მყოფ ელემეტზე.

VectorNth მეთოდის მუშაობა უნდა აფერხებდეს VectorReplace და VectorInsert მეთოდებს მხოლოდ და მხოლოდ მაშინ, როცა VectorNth ფუნქცია მიმართავს უფრო მარჯვენა პოზიციის მქონე ელემენტს ვიდრე დანარჩენი ორი.

მესამე პუნქტი უნდა სრულდებოდეს VectorReplace ფუნქციისთისაც.

```
typedef void (*VectorFreeFunction)(void *elemAddr);

typedef struct {
        void * elems;
        int elemSize;
        int loglen;
        int alloclen;
        VectorFreeFunction freeFn;



} vector;


void *VectorNth(const vector *v, int position){

        assert(position >= 0 && position < v->loglen);

        return (void*)((char *)v->elems + position*v->elemSize);
}

void VectorReplace(vector *v, const void *elemAddr, int position){

        assert(position >= 0 && position < v->loglen);

        char * address = VectorNth(v,position);

        if(v->freeFn!=NULL) v->freeFn(address);

        memcpy(address,elemAddr,v->elemSize);

}
void VectorInsert(vector *v, const void *elemAddr, int position){

        assert(position>=0 && position<=v->loglen);

        if(v->loglen==v->alloclen) Grow (v);

        size_t length = (v->loglen-position)*v->elemSize;

        v->loglen++;

        char * address = VectorNth(v,position);

        memmove(address+v->elemSize,address,length);

        memcpy(address,elemAddr,v->elemSize);

}

void VectorDelete(vector *v, int position){

        assert(position >= 0 && position < v->loglen);
```

```
        char * address = VectorNth(v,position);
        if(v->freeFn!=NULL) v->freeFn(address);
        size_t length = (v->loglen-position-1)*v->elemSize;
        memmove(address, address+v->elemSize,length);
        v->loglen--;
}

static void Grow(vector *v){
        v->alloclen = 2*v->alloclen;
        v->elems = realloc (v->elems, v->alloclen * v->elemSize);
        assert(v->elems!=NULL);
}
```

ამოცანა 27.(2013 წლის შუალედური) ეპილოგი (15 ქულა)
ა) რას დაბეჭდავს: ძველი
ბ) მოიყვანეთ მაგალითი როდესაც შედარების ფუნქცია:

```
int cmpfn(int a, int b){
        return (a-b);
}
```

იმუშავებს არასწორად.

ამოცანა 28.(2013 წლის შუალედური) დუბლიკატები (20 ქულა)
თქვენი ამოცანაა დაწეროთ ფუნქცია HasDublicates რომელსაც გადაეცემა C-სტრგინგების მასივი და რომელიც ამოწმებს ამ მასივში არის თუ არა გამეორებული სტრინგი. სტრინგები რომლებიც იგივე სიმბოლოებისაგან შედგება არ ითვლება გამეორებულად. გვაინტერესებს მხოლოდ ის შემთხვევა როდესაც მეორდება მიმთითებელი. ამ ამოცანის ამოხსნისთვის აუცილებელია გამოიყენოთ ფუნქცია lfind განსხვავებული მიდგომის შემთხვევაში ამოცანა ამოხსნილად არ ჩაითვლება.
ფუნქციამ უნდა დააბრუნოს true თუკი დუბლიკატი მოიძებნა, false - წინააღმდეგ შემთხვევაში.
შეგახსენებთ lfind-ის ჰედერს:

```
void *lfind(const void *key, const void *arr, size_t *nElems, size_t elemSize, int (*cmp)(const void *, const void *));
bool HasDublicates(char *a[], int n){
```

ამოცანა 30.(2013 წლის ფინალური) რას დაბეჭდავს? (20 ქულა)

რას დაბეჭდავენ შემდეგი C++ პროგრამები? თუკი პროგრამის კომპილაციის ან მსვლელობის დროს მოხდება შეცდომა დაწერეთ თუ რა შეცდომაა და აღწერეთ თუ რატომ ხდება. გაითვალისწინეთ ის ფაქტი რომ არსებობენ სხვადასხვა სისტემები(big endian/small endian, 32bit/64bit).
ა) ძველი

ბ)
```
#include<iostream>
using namespace std;
class Test{
private:
  int x;
  int y;
public:
  Test(int x = 0, int y = 0) { this->x = x; this->y = y; }
  Test setX(int a) { x = a; return *this; }
  Test setY(int b) { y = b; return *this; }
  void print() { cout<< "x = " <<x<< " y = " <<y<< endl; }
};
int main()
{
  Test obj1;
  obj1.setX(10).setY(20);
  obj1.print();
  return 0;
}
```
პასუხი:

ამოცანა 31.(2013 წლის გადაბარება) სპელჩეკერი (30 ქულა)
თქვენი მიზანია დაწეროთ სპელჩეკერი. ამისათვის თქვენ უნდა დაწეროთ Spellcheck ფუნქცია რომელსაც გადაეცემა ტექსტი და ლექსიკონი და რომელიც გადაცემულ ტექსტს გაასწორებს. ლექსიკონი დალაგებული სი სტრინგების მასივია. აუცილებელი მოთხოვნაა ლექსიკონში სიტყვის ძებნისას გამოიყენოთ ფუნქცია bsearch. შეგახსენებთ ფუნქციის ჰედერს:
void * bsearch(const void *key, const void *base, size_t nel, size_t width, int (*compar) (const void *, const void *));
ტექსტის გასწორებისას ჩათვალეთ რომ სიყტვები ერთმანეთისგან გამოყოფილია პრაბელებით და ტექსტში ინგლისური ასოებისა და პრაბელების გარდა არაფერი არ გვაქვს. ასევე ჩათვალეთ რომ ყოველ შეცდომით დაწერილ სიყტვაში ერთადერთი ტიპის შეცდომაა - მასში რომელიღაც ასო არის გამოტოვებული. ამასთან შეგვიძლია დავუშვათ რომ სიტყვის გასწორება ერთადერთი ხერხით შეიძლება ანუ ლექსიკონში არსებობს ერთადერთი სიტყვა რომელსაც ერთერთ ასოს თუ ამოვაკლებთ მივიღებთ ჩვენს შეცდომით დაწერილ სიტყვას.

```
void Spellcheck(char * text, char* [] lex){
```

ამოცანა 32. (2013 წლის გადაბარება,2015 წლის გადაბარება) რას დაბეჭდავს? (20 ქულა)

რას დაბეჭდავენ შემდეგი C პროგრამები? პასუხი დაასაბუთეთ. თუკი პროგრამის კომპილაციის ან მსვლელობის დროს მოხდება შეცდომა დაწერეთ თუ რა შეცდომაა და აღწერეთ თუ რატომ ხდება. გაითვალისწინეთ ის ფაქტი რომ არსებობენ სხვადასხვა სისტემები(big endian/small endian, 32bit/64bit).

ა)

```
#include <stdio.h>
int main()
{
 float a = 0;
 printf("%d\n", a);
 printf("%d\n", *(int *)&a);
 return 0;
}
```

პასუხი:

ბ)

```
#include <stdio.h>
int main()
{
	int i=543;
	printf("%d\n",printf("%d",printf("%d",i)));
	return 0;
}
```

პასუხი:

ამოცანა 33.(2013 წლის გადაბარება) ამოცანები ნაკადებზე(50 ქულა)

ა) დამბეჭდე ყოვლისშემძლე მაჯლაჯუნავ

მოცემული გაქვთ N ელემენტიანი მთელი რიცხვების მასივი და M ცალი ნაკადი. თითოეული ნაკადი იძახებს print მეთოდს. თქვენი ამოცანაა დაწეროთ print მეთოდის რეალიზაცია, ისე რომ რიცხვები დაიბეჭდოს იგივე მიმდევრობით როგორც მასივშია ჩაწერილი. ამავდროულად პირველი რიცხვი უნდა დაბეჭდოს პირველმა ნაკადმა, მეორე მეორემ და ასე შემდეგ, M+1 რიცხვი უნდა დაბეჭდოს ისევ პირველმა ნაკადმა, M+2 მეორემ.

```
#define M ???
#define N ???

static int numbers[N];

void * print(void * i) {
```

```
}

int main() {

        for(i=0;i<M;i++){
                pthread_t t;
                pthread_create(&t, NULL, print, (void*)i);
        }

}
```

ბ) ყველამ ერთად

ქვემოთ მოცემული კოდში ნათლად ჩანს, რომ N ცალი ნაკადი იძახებს work მეთოდს, რომელიც აგენერირებს შემთხვევით რიცხვს და რაღაც დროის შემდეგ იძახებს print_res მეთოდს. print_res მეთოდი ჯამავს ყველა ნაკადის მიერ დაგენერირებულ რიცხვებს და ბეჭდავს. შეცვალეთ არსებული კოდი ისეთნაირად, რომ print_res მეთოდი ელოდებოდეს სანამ ყველა ნაკადი არ გამოიძახებს მას და მხოლოდ ამის შემდეგ რომელიმე ერთი ჯამავდეს ყველა დაგენერირებულ რიცხვს და ბეჭდავდეს. დაბეჭდვის შემდეგ ყველა ნაკადი უნდა აგრძელებდეს ჩვეულებრივ მუშაობას.

```
#define N ???

static int data[N];


void print_res() {
   int total = 0;
   for (int i=0; i<N; i++) {
      total += data[i];
   }

   printf("total=%d", total);
}
```

```
void work(int indx) {
        while (1) {
           data[indx] = rand();
           int t = rand();
           sleep(t);
           print_res();
        }
}

int main() {

        for(i=0;i<N;i++){
                pthread_t t;
                pthread_create(&t, NULL, work, (void*)i);
   }

}
```

ამოცანა 36.(2014 წლის შუალედური) ეპილოგი (15 ქულა)
რას დაბეჭდავენ შემდეგი პროგრამები? მიუთითეთ კომპილაციისას ამოგდებული ვორნინგები. გაითვალისწინეთ კომპიუტერის ყველა სტანდარტული კოფიგურაცია(bigendian, littleendian, 64bit, 32bit). თუკი მოხდება კომპილაციის შეცდომა ახსენით რატომ.

ა)

```
#define Bla(A,B) A + B
int main(){
  int * x = 20;
  int * y = 30;
  printf("%d", (int)x * (int)Bla(x,y));
}
```

ბ)

```
#include <stdio.h>
int main(){
  char str[10];
  str[0] = 'a';
  str[1] = 'z';
  str[2] = '\0';
  str++;
  printf("%s", str);
}
```

გ)

```
#include <stdio.h>
int main(){
  char str[10];
  str[0] = 'a';
  str[1] = 'z';
  str[2] = '\0';
  char * newStr = str+1;
  printf("%s", newStr);
}
```

ამოცანა 37.(2014 წლის შუალედური) მასივიდან ბმული სია (15 ქულა)
მოცემული გაქვთ elemsize სიგრძის ელემენტები, მოცემული გაქვთ მასივში ელემენტების რაოდენობა და მოცემული გაქვთ მასივზე მიმთითებელი. საჭიროა მასივის კოპიოს გაკეთება ჰიპში. მაგრამ ჰიპი არის დაჩეხილი უკვე გამოყენებული ადგილებით და არ არის საკმარისი ერთმანეთზე მიყოლებული ადგილი რათა ზუსტად იგივენაირი მასივი გაკეთდეს(malloc(elemsize*n) აბრუნებს NULL-ს). სამაგიეროდ ბმული სიისთვის საკმარისი ადგილი მოიძებნება. თქვენი მიზანია დაწეროთ arr_to_list ფუნქცია რომელიც დააბრუნებს ბმული სიის სათავეზე მიმთითებელს.
void *arr_to_list(void *arr, int nelems, int elemsz)

ამოცანა 38.(2014 წლის შუალედური) თავისუფალი ადგილი(15 ქულა)
მოცემული გაქვთ მიმთითებელი რომელიც დააბრუნა malloc ფუნქციამ. თქვენი მიზანია გაიგოთ თუ რამდენი ბაიტია თავისუფალი ადგილი ამ მიმთითებლიდან მანამ სანამ სხვა malloc-ის მიერ გამოყოფილ ადგილს არ შევეჯახებით. ანუ რამდენი ბაიტის გამოყენება შეუძლია realloc-ს საწყისი მიმთითებლის შეცვლის გარეშე. ფუნქციის შესრულების შემდეგ თუ როგორი იქნება მეხსიერების მდგომარეობა მნიშვნელობა არ აქვს.
int free_space_from_pointer(void * e){

ამოცანა 39.(2014 წლის შუალედური) replace (15 ქულა)
თქვენი ამოცანაა დაწეროთ replace ფუნქციის რეალიზაცია. მოცემული გაქვთ generic მასივი elemsize ზომის ელემენტებით. მოცემული გაქვთ ელემენტი რომლის შეცვლაც არის საჭირო. მასივში რეალურად ყველა შემხვედრი ეს ელემენტი უნდა შეიცვალოს. ასევე მოცემული გაქვთ ახალი ელემენტი რომლითაც ხდება ძველი ელემენტის შეცვლა. მაგალითად თუ გვაქვს მასივი {1,2,2,1} ძველი ელემენტი 1 და ახალი ელემენტი 3 მაშინ უნდა მივიღოთ შემდეგი მასივი {3,2,2,3}.

```
/*
arr             - ჯენერიკ მასივზე მიმთითებელი
elemnum         - მასივში ელემენტების რაოდენობა
elemsize                - მასივში შენახული ელემენტების ზომა
old             - მისამართზე ინახება ელემენტი, რომლის ტოლი ელემენტებიც უნდა
შეიცვალოს
new             - მისამართზე ინახება ელემენტი რომლითაც უნდა შეიცვალოს
cmp             - ელემენტების შედარების ფუნქცია
```

```
*/
void replace(     void * arr,
                  const size_t elemNum,
                  const size_t elemSize,
                  void * old,
                  void * new,
                  int (* cmp) (void *, void *)) {



}
```

replace ფუნქცია arr მასივში არსებულ ყველა ელემენტს რომელიც old მისამართზე შენახული ელემენტის ტოლია, უნდა ცვლიდეს new მისამართზე შენახული ელემენტით. ჩათვალეთ რომ ყველა ელემენტის ზომა როგორც old-ის ისე new-ის არის elemsize.

ამოცანა 40.(2014 წლის შუალედური) სტრინგების replace(30 ქულა).
თქვენი ამოცანაა replace ფუნქცია(იხილეთ ამოცანა 4) გამოიყენოთ სტრინგების ჩანაცვლებისთვის. დაწერეთ შემდეგი ფუნქციის იმპლემენტაცია:

```
/*
strs      - მეხსიერება სადაც ინახება ერთამენთის მიმდევრობით ჩაწერილი c სტრინგები
strsNum           - strs მისამართზე ჩაწერილი სტრინგების რაოდენობა
old       - სტრინგი რომელიც უნდა შეიცვალოს
new       - სტრინგი რომლითაც უნდა შეიცვალოს
return    - განახლებული მეხსიერების მისამართი
*/
void * strReplace(char * strs, size_t strsNum, char *old, char *new) {


}
```

strs მისამართზე ერთმანეთის მიმდევრობით ჩაწერილია c სტრინგები, ყოველი სტრინგის ბოლოს გვხვდება სიმბოლო ‘\0’ რომლის შემდეგაც იწყება მომდევნო სტრინგი.
size_t strNum - ცვლადში ინახება სტრინგების რაოდენობა.

ადვილად წარმოსადგენად ქვემოთ მოცემულია strs მეხსიერების ერთერთი შესაძლო მდგომარეობა, სადაც 4 სტრინგია ჩაწერილი:

string1\0string2\0string3\0string4\0

თქვენი მიზანია დაწეროთ strReplace ფუნქციის იმპლემენტაცია მე 4 ამოცანაში განსაზღვრული replace ფუნქციის გამოყენებით. strReplace ფუნქცია უნდა აბრუნებდეს მისამართს სადაც ინახება strs მისამართზე არსებული სტრინგები(იგივე ფორმატით), იმ განსხვავებით რომ ყველა old სტრინგი შეცვლილი იქნება new სტრინგზე.

მაგალითები:

```
strs              - "aabbcc\0aa\0bb\0aa\0"
strsNum           - 4
old               - "aa\0"
new               - "bbccdd\0"
```

უნდა დააბრუნოს მეხსიერების მისამართი სადაც ჩაწერილია "aabbcc\0bbccdd\0bb\0bbccdd\0"

strs - "aabbcc\0aa\0bb\0aa\0"
strsNum - 4
old - "aabb\0"
new - "bbccdd\0"

უნდა დააბრუნოს მეხსიერების მისამართი სადაც ჩაწერილია "aabbcc\0aa\0bb\0aa\0"

ამოცანა 42.(2014 წლის ფინალური) სპელკორექტორი (30 ქულა)
თქვენი მიზანია სპელკორექტორის დაწერა ამისათვის გაქვთ ლექსიკონი რომელშიც ყველა სიტყვა წერია. უნდა დაწეროთ ფუნქცია რომელსაც გადმოეცემა ლექსიკონი და რაღაც წინადადება და რომელიც ამ წინადადებას გადააქცევს სწორი, გამართული სიტყვებისგან შედგენილ წინადადებად.
ჩათვალეთ რომ:
1. ლექსიკონი არის C-სტრიგნგების ზრდადობით დალაგებული მასივი.
2. წინადადება არის C-სტრინგი რომელიც შედგება მხოლოდ პატარა ასოებისა და სფეისებისგან.
3. წინადადებაში არსებული სიტყვები ერთმანეთისგან გამოყოფილია ერთი სფეისით.
4. შეცდომები რომელიც წინადადებაში შეიძლიება იყოს არის მხოლოდ და მხოლოდ ორი სახის. პირველი რომელიმე გამართულ სიტყვას ბოლოში აქვს მიწერილი ერთი სიმბოლო(რის შედეგადაც ის სიტყვა წერია შეცდომით) ან მეორე - წინადადებაში რომელიმე გამართულ სიტყვას დააკლეს ბოლო ერთი სისმბოლო. სხვა ტიპის შეცდომები წინადადებაში არ გვხვდება.

აუცილებელია რომ:
1. თქვენი პროგრამა მუშაობდეს ეფექტურად - O(logN)*m დროში სადაც N ლექსიკონში არსებული სიტყვების რაოდენობაა ხოლო m კი წინადადებაში არსებული სიტყვების რაოდენობა.
2. გამოიყენოთ სტანდარტული ორობითი ძებნის ფუნქცია და არ დაწეროთ თქვენი.
void * bsearch(const void *key, const void *base, size_t nel, size_t width,int (*compar) (const void *, const void *));
3. მეთოდის დასრულების შემდეგ sent უნდა მიუთითებდეს მეხსიერებაზე სადაც გასწორებული წინადადება წერია.

void spellCorrect(char*[] lex, int n, char** sent){

ამოცანა 43.(2014 წლის ფინალური) რას დაბეჭდავს? (10 ქულა)

რას დაბეჭდავს შემდეგი C პროგრამა? თუკი პროგრამის კომპილაციის ან მსვლელობის დროს მოხდება შეცდომა დაწერეთ თუ რა შეცდომაა და აღწერეთ თუ რატომ ხდება. გაითვალისწინეთ ის ფაქტი რომ არსებობენ სხვადასხვა სისტემები(big endian/small endian, 32bit/64bit).

```c
#include <stdio.h>

void a(int k, int p) {
  int *arr = (int*)&arr;//ჩათვალეთ  რომ &arr არის  1000
  printf("%d %d %d %d\n",*(arr-1),*(arr-2),*(arr-3),*(arr-4));
}

int main()
{
  a(1, 9);
  printf("Hello, World!\n");
  return 0;
}
```

პასუხი:

ამოცანა 44.(2014 წლის ფინალური) ლიფტი(60 ქულა)

ერთერთ საცხოვრებელ სახლში ცხოვრობს N ადამიანი, კორპუსს აქვს M სართული და მხოლოდ ერთი ლიფტი.
მეზობლები ხშირად დადიან სტუმრად ერთმანეთთან და თან გადაადგილებისთის ყოველთვის ლიფტს იყენებენ.

თქვენი ამოცანაა დაწეროთ პროგრამა რომელიც მოახდენს ლიფტის გადაადგილების სიმულირებას, შემდეგი პირობების გათვალისწინებით:
1. ლიფტი თავდაპირველად დგას ნულოვან სართულზე და ელოდება გამოძახებას, 2 წამში ერთხელ ამოწმებს არის თუ არა გამოძახება.
2. როგორც კი ვინმე გამოიძახებს ლიფტს, იგი იწყებს მოძრაობას და სართულიდან სართულზე გადასვლას 2 წამს ანდომებს.
3. ბინის მაცხოვრებლები მთელი დღის განმავლობაში გადაადგილდებიან ერთი სართულიდან მეორეზე.
4. ერთი სართულიდან მეორეზე გადასვლის დროს მგზავრი პირველ რიგში იძახებს ლიფტს სართულზე სადაც თვითონ იმყოფება(CallButton ფუნცია), ლიფტის მოსვლის შემდეგ შედის ლიფტში და იძახებს დანიშნულების სართულს(ToButton ფუნქცია). როგორც კი ლიფტი გაჩერდება დანიშნულები სართულზე მგზავრი ტოვებს ლიფტს.
5. თუ მგზავრი იმყოფება ლიფტში მას არ უნდა შეეძლოს ლიფტის გარედან გამოძახება(CallButton).
6. ლიფტი მოძრაობს სტანდარტული ალგორითმით, პირველი გამოძახების შემდეგ იწყებს სვლას შესაბამისი სართულისკენ, თან 2 წამში ერთხელ ამოწმებს ხომ არ გაჩნდა ახალი გამოძახებები.
ლიფტი აგრძელებს საწყისი მიმართულებით სვლას მანამ სანამ არ მიაღწევს კიდურა წერტილს(ჩასხდომის ან ჩამოსვლის), გზადაგზა ჩერდება ყველა იმ სართულზე რომელზე გაჩერებაც მოითხოვეს მგზავრებმა(ჩასხდომის ან ჩამოსვლის მიზნით). კიდურა წერტილზე მიღწევის მომენტში თუ ლიფტს აქვს სხვა გამოძახებები, მაშინ იგი იწყებს მოძრაობას საპირისპირო მიმართულებით იგივე ალგორითმით. ქვემოთ მოყვანილია ლიფტის მოძრაობის მაგალითები.

თითოეულ სტრიქონში მოპცემულია ლიფტის მოქმედებები 2 წამის ინტერვალით.

0 წამზე - CallButton(Floor 5)
2 წამზე - ElevatorUp(Floor 1)
4 წამზე - ElevatorUp(Floor 2) & CallButton(Floor 4)
6 წამზე - ElevatorUp(Floor 3) & CallButton(Floor 1)
8 წამზე - ElevatorUp(Floor 4) & ElevatorStop(Floor 4) & ToButton(Floor 2)
10 წამზე - ElevatorUp(Floor 5) & ElevatorStop(Floor 5) & ToButton(Floor 6)
12 წამზე - ElevatorUp(Floor 6) & ElevatorStop(Floor 6)
14 წამზე - ElevatorDown(Floor 5)
16 წამზე - ElevatorDown(Floor 4)
18 წამზე - ElevatorDown(Floor 3)
20 წამზე - ElevatorDown(Floor 2) & ElevatorStop(Floor 2)1
22 წამზე - ElevatorDown(Floor 1) & ElevatorStop(Floor 1)

```
#define PEOPLE_NUM 10;
#define FLOOR_NUM 10;

typedef struct {

} elevator;

typedef struct {
	int floorNumber;//ამ მომენტში რომელ სართულზე იმყოფება მგზავრი
	int idNumber;//მგზავრის იდენტიფიკატორი

} person;


void ElevatorStop(elevator *e) {

}

void ElevatorUp(elevator *e) {

}

void ElevatorDown(elevator *e){

}

void CallButton(int onFloor) {

}


void * ToButton(int toFloor) {

}



void * PersonLife(void * data) {
	person * p = (person *)data;
	while (true) {
		int pauseTime = rand()%10;
		pause(pauseTime);
		int toFloor = rand() % FLOOR_NUM;
		CallButton(p.floorNumber);
		ToButton(toFloor);
	}
```

```
}


void * MoveElevator(void * data) {
	while (true) {
		pause(2);
		//აქ უნდა დაწეროთ ლიფტის მოძრაობის ლოგიკა,
//დეკომპოზიციისთვის შეგიძლიათ გამოიყენოთ ElevatorStop,
//ElevatorUp და ElevatorDown მეთოდები.
	}
}



int main() {
	elevator e;
	pthread_t elevatorThread;
	pthread_create(&elevatorThread, NULL, MoveElevator, &e);

	person persons[PEOPLE_NUM];
	for (int i=0; i<PEOPLE_NUM; i++) {
		pthread_t t;
		persons[i].idNumber = i;
		persons[i].floorNumber = i;
		pthread_create(&t, NULL, PersonLife, persons+i);

	}
	pthread_join(&elevatorThread);
	pthread_exit(NULL);

}
```

ამოცანა 47.(2014 წლის გადაბარება) თამაში(40 ქულა)
თქვენი მიზანია გააუმჯობესოთ ბრაუზერის მუშაობა. მოცემული გაქვთ makeHttpRequest და processAllHttpsRequeusts ფუნქციები, რომელთა გამოყენება ხდება შემდეგი ინსტრუქციების მიხედვით:

M ცალი ნაკადიდან მუდმივად ხდება makeHttpRequest ფუნქციის გამოძახება. makeHttpRequest ფუნქცია urlData რიგში ამატებს გადმოცემულ სტრუქტურას, იბლოკება მანამ სანამ processAllHttpsRequeusts ფუნქცია არ ჩამოტვირთავს შესაბამის მონაცემებს და შემდეგ იძახებს displayData მეთოდს.
processAllHttpsRequeusts ფუნქციის გამოძახება მუდმივად ხდება მხოლოდ ერთი ნაკადიდან, იგი იბლოკება სანამ რიგში დამატებული ელემენტების რაოდენობა N-ზე ნაკლებია. როგორც კი რაოდენობა გახდება N ის ტოლი, ან მეტი, ფუნქცია რიგრიგობით იწყებს მონაცემების ჩამოტვირთვას სანამ რიგი არ გაცარიელდება.
თქვენი ამოცანაა დაასრულოთ makeHttpRequest და processAllHttpsRequeusts ფუნქციების რეალიზაცია, ისე რომ სისტემამ გამართულად იმუშაოს

```
struct data {
	char * url;
	void * data;
	int dataSize;


}
/* global shared data goes here */
queue urlData;  // supports size(), push(), pop(), and any other queue
// operations you need



makeHttpRequest(data *d) {

	urlData.push(d);


displayData(d);  //უზრუნველყოფს d->data ში წაკითხული მონაცემების
//ვიზუალიზაციას


}

processAllHttpsRequeusts() {

	/* service all the requests in the queue */
	while (urlData.size() != 0) {


		data* d = urlData.pop();


		readHttpRequest(d);	//უზრუნველყოფს d->url დან მონაცემების
// წაკითხვას და d->data ში ჩაწერას


	}


}
```

ამოცანა 48.(2015 წლის შუალედური) ეპილოგი (15 ქულა)
რას დაბეჭდავენ შემდეგი პროგრამები? გაითვალისწინეთ კომპიუტერის ყველა სტანდარტული კონფიგურაცია(big endian, small endian, 32 bit, 64 bit). თუკი მოხდება კომპილაციის შეცდომა ახსენით რატომ. თუკი პროგრამა დაიქრეშება ახსენით რის გამო და ასევე ახსენით რა შეცდომას ამოაგდებს. ჩათვალეთ რომ არც ერთ პროგრამას include-ები არ გააჩნია.

ა)
```
main(){
        finish(start());
}
int start(){
        int x = 4;
        printf("%d ", x);
        return x;
}
void finish(int z){
        short x;
        short y;
        printf("%d %d %d", x, y, z);
}
```

ბ)
```
main(){
        start()
        finish();
}
void start(){
        int x = 4;
}
void finish(){
        short x;
        short y;
        printf("%d %d", x, y);
}
```

გ)
```
int main() {
int* pointer = (int*)malloc(sizseof(int));
*pointer = 10;
printf("%d", *pointer)
}
```

ამოცანა 49.(2015 წლის შუალედური) სუ კაი სტუდენტები (15 ქულა)
მეხსიერებაში ერთმანეთის მიმდევრობით ჩაწერილია სტუდენტების სახელები და ქულები. სახელი ჩაწერილია C სტრინგის საშუალებით რომელიც ბოლოვდება '\0' სიმბოლოთი, ხოლო ქულა ჩაწერილია int ტიპის მნიშვნელობით, იხ. სურათი:

Nozadze\0	85	Otiashvili\0	95	Janashia\0	59	Gogoli\0	91

დაწერეთ printA ფუნცქია, რომელსაც გადაეცემა ზემოთ აღწერილი მეხსიერების მისამართი და რომლემაც უნდა დაბეჭდოს ყველა იმ სტუდენტის სახელი რომლის

ქულაც 91 ზე მეტი ან ტოლია. გაითვალისწინეთ რომ ქულა შეიძლება ეწეროს არა 4- ის ჯერად მისამართზე.

```
void printA(void * data) {




}
```

ამოცანა 50.(2015 წლის შუალედური) მეხსიერება (60 ქულა)
თქვენი ამოცანაა დაწეროთ მეხსიერების მართვის ფუნქციონალი, რომელიც სპეციალურად გამოყოფილ ადგილს გამოიყენებს როგორც ჰიპ მეხსიერებას.
Heap სტრუქტურის საშუალებით მოცემული გაქვთ ვირტუალური ჰიპის აღწერა, უნდა დაწეროთ malloc, free, realloc, memcpy და memove ფუნქციების ანალოგიური ფუნქციები.

```
typedef struct {
        void * heap;
        size_t heapSize;
        void * freeList;
} Heap;
```

დაწერეთ შემდეგი ფუნქციების რეალიზაცია:

```
void *    cmalloc(heap* vm, size_t size);
void      cfree(heap * vm, void *ptr);
void *    crealloc(heap * vm, void *ptr, size_t size);
void *    cmemcpy(heap * vm, void *dest, const void *src, size_t n);
void *    cmemmove(heap * vm, void *dest, const void *src, size_t n);
```

თითოეული მათგანი უნდა იყენებდეს მხოლოდ vm მისამართზე გამოყოფილ მეხსიერებას.
heap მისამართზე გამოყოფილი მეხსიერება შეგიძლიათ გამოიყენოთ როგორც ვირტუალური ჰიპი. მოცემული გაქვთ ასევე ჰიპის ზომა. ამ მეხსიერების გარეთ წვდომა არ გაქვთ და შესაბამისად cmalloc და crealloc ფუნქციები უნდა აბრუნებდნენ მხოლოდ [heap, (char*)heap+heapSize] შუალედში არსებული მეხსიერების მისამართს, რომელზეც შესაძლებელი იქნება სხვადასხვა ტიპის ცვლადების მნიშვნელობების ჩაწერა წაკითხვა.
freeList მისამართზე გამოყოფილი მეხსიერება შეგიძლიათ გამოიყენოთ, მხოლოდ მეხსიერების მენეჯმენტისთვის საჭირო ინფორმაციის შესანახად, როგორიცაა freelist-ები მაგალითად, ჩათვალეთ რომ freeList მისამართზე საკმარისად დიდი ადგილია გამოყოფილი იმისთვის რომ ნებისმიერი ზომის freelist შეინახოთ. შეგახსენებთ რომ freelist არის სტრუქტურა რომელიც გეხმარებათ იმის გარკვევაში თუ რომელი ადგილებია ჰიპში ცარიელი და რომლებია დაკავებული.

cmalloc - ფუნქციას გადაეცემა ბაიტების რაოდენობა და აბრუნებს მეხსიერების მისამართს [vm->heap, (char*) vm->heap+heapSize] შუალედიდან. თუ ვირტუალურ მეხსიერებაში არ არის სამაირის ადგილი, ფუნქციამ უნდა დააბრუნოს NULL ნიშვნელობა.

cfree - გადაეცემა მხოლოდ cmalloc ის ან crealloc ის მიერ დაბრუნებული მისამართი. ფუნქციამ vm სტრუქტურაში უნდა მონიშნოს, რომ მითითებულ მისამართზე დაკავებული მეხსიერება განთავისუფლდა, რის შემდეგაც cmalloc და crealloc ფუნქციებს მიეცემათ საშუალება გამოიყენონ აღნიშნული მეხსიერება.

crealloc - ფუნქციას გადაეცემა cmalloc ის ან crealloc ის მიერ დაბრუნებული მისამართი და ბაიტების რაოდენობა სადამდეც გვინდა რომ გაიზარდოს დაკავებული მეხსიერების ზომა. ფუნქციამ უნდა დააბრუნოს მისამართი სადაც size ზომის მეხსიერება არის გამოყოფილი. თუ საკმარისი მეხსიერება არ არის თავისუფალი, ფუნქციამ უნდა დააბრუნოს NULL.

cmemcpy - ფუნქციამ src დან dest მისამართზე უნდა გადააკოპიროს n ბაიტ ინფორმაცია. ფუნქცია აბრუნებს src მისამართს ან NULL ს თუ გადაკოპირება ვერ ხერხდება.

cmemmove - ფუნქციამ src დან dest მისამართზე უსაფრთხოდ უნდა გადააკოპიროს n ბაიტ ინფორმაცია. src და dest მეხსიერებები შესაძლოა ერთმანეთს კვეთდნენ, ამიტომ cmemmove ფუნქცია კოპირებისთვის სარეზერვო მეხსიერებას უნდა იყენებდეს. ფუნქცია აბრუნებს src მისამართს ან NULL ს თუ გადაკოპირება ვერ ხერხდება.

ამოცანა 52.(2015 წლის ფინალური) მაქსიმალური სიგრძის სტრინგი (20 ქულა)
თქვენი ამოცანაა C-სტრინგების მასივში მოძებნოთ ყველაზე გრძელი სტრინგი, ოღონდ ისე რომ არ გამოიყენოთ ციკლი და რეკურსია. სამაგიეროდ უნდა გამოიყენოთ lsearch ფუნქცია.
შეგახსენებთ lsearch ფუნქციის ჰედერს:

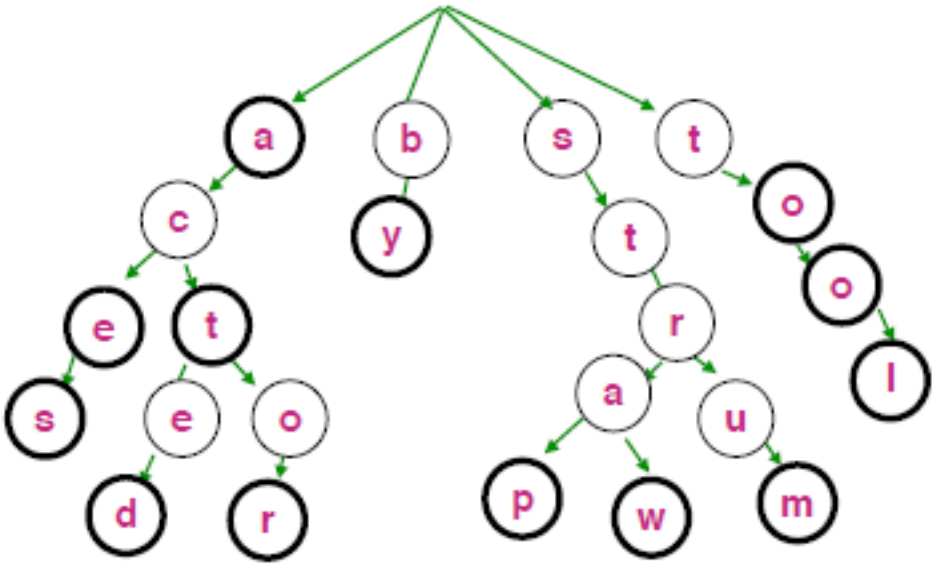
```
void * lsearch(const void *key, void *base, size_t *nelp, size_t width,
                                int (*compar)(const void *, const void *));
```

დაწერეთ ფუნქცია longestString, რომელსაც გადაეცემა სტრინგების მასივზე მიმთითებელი და რომელიც აბრუნებს ყველაზე გრძელი სიტყვის სიგრძეს.

```
int longestString(char ** stringArray){
```

ამოცანა 53.(2015 წლის ფინალური) Trie ხე (40 ქულა)

თქვენ ამოცანაა დაწეროთ “Trie” სტრუქტურის იმპლემენტაცია, სურათზე შეგიძლიათ იხილოთ როგორ გამოიყურება სტრუქტურა, გამუქებული რგოლებით ნაჩვენებია Node-ები რომლებზეც სრულდება სიტყვა.



(მოცემულ Trie სტრუქტურაში ინახება შემდეგი სიტყვები: a, ace, aces, act, acted, actor, by, strap, straw, strum, to, too, tool)

Trie სტრუქტურა წარმოადგენს ხეს, რომელიც ინახავს სიტყვების სიმრავლეს, ისე რომ ერთი და იგივე პრეფიქსის მქონე სიტყვები იყენებენ ერთი და იგივე Node-ებს, როგორც სურათზეა ნაჩვენები.
თქვენი ამოცანაა მოიფიქროთ Trie ხის სტრუქტურა და დაწეროთ შემდეგი მეთოდების რეალიზაცია:

```
typedef bool int;
#define true 1
#define false 0

/*სიტყვის დამატება Trie სტრუქტურაში*/
void add(char * w) {

}

/*სიტყვის წაშლა Trie სტრუქტურიდან, სიტყვის წაშლა გულისხმობს ყველა უსარგებლო Node ის წაშლას.
თუ სიტყვა საერთოდ არ გვხვდება ლექსიკონში, მეთოდი არაფერს არ აკეთებს*/
void remove(char * w) {

}

/*აბრუნებს true-ს თუ გადაცემული სიტყვა ინახება სტრუქტურაში, წინააღმდეგ შემთხვევაში false-ს*/
bool containsWord(char * w) {


}

/*აბრუნებს true ს თუ სტრუქტურაში ინახება სიტყვა რომლის პრეფიქსიც ემთხვევა გადაცემულ სტრინგს*/
bool containsPrefix(char * w) {

}
```

ამოცანა 54.(2015 წლის ფინალური) ბანკი(50 ქულა)

თქვენი ამოცანაა დაწეროთ ბანკის სერვის ცენტრში კლიენტების მომსახურების სიმულაცია. სერვის ცენტრში მუშაობს ოთხი ოპერატორი და ერთი მოლარე. კლიენტი ბანკში მისვლისას იღებს რიგის ნომერს და ელოდება თავის რიგს. თავისუფალი ოპერატორი რიგის ნომრის მიხედვით იძახებს მომდევნო კლიენტს და ემსახურება, შემდეგ ამისამართებს კლიენტს მოლარესთან, თვითონ კი იძახებს მომდევნო კლიენტს და ასე გრძელდება დღის ბოლომდე.
მოლარე, ოპერატორის ანალოგიურად, იძახებს კლიენტებს რიგის მიხედვით და ემსახურება, იმ განსხვავებით რომ მოლარის რიგი დგება არა რიგის ნომრის მიხედვით, არამედ იმის და მიხედვით თუ როდის მოხდა კლიენტის გადამისამართება ოპერატორიდან.

ჩათვალეთ რომ RunClientTests(), WorkDayIsNotEnd(),CashierServClient(),OperatorServClient() მეთოდები იმპლემენტირებულია. თქვენ უნდა დაწეროთ დანარჩენი კოდის რეალიზაცია ისე რომ სიმულაციამ სწორად იმუშაოს. შეგიძლია უკვე დაწერილ კოდში ცვლილებებიც გააკეთოთ. თქვენი მოსაფიქრებელია ასევე Client სტრუქტურის შინაარსიც რომელიც Client მეთოდს გადაეცემა.

გაითვალისწინეთ რამდენიმე შენიშვნა:
თუ რიგში არავინ არ დგას ოპერატორის ნაკადი უნდა იყოს მოლოდინის რეჟიმში.
თუ რიგში არავინ არ დგას მოლარის ნაკადი უნდა იყოს მოლოდინის რეჟიმში.
როცა კლიენტი ელოდება ტავის რიგს, კლიენტის ნაკადიც უნდა იყოს მოლოდინის რეჟიმში.
არ გაქვთ Busy waiting ის გამოყენების უფლება
შეგიძლიათ შემოიტანოთ თქვენთვის მოსახერხებელი სტრუქტურები და გლობალური ცვლადები

```
#include "helper.c"

#define OPERATOR_NUM 4;

void * Cashier(void * data) {
	while (WorkDayIsNotEnd()) {

		CashierServClient();

	}
}

void * Operator(void * data) {
	while (WorkDayIsNotEnd()) {

		OperatorServClient();

	}
}

void * Client(void * data) {
```

```
Client * c = (Client *)data;
//კლიენტმა უნდა მიიღოს რიგის ნომერი და დაელოდოს თავის რიგს, გამოძახების შემდეგ მივიდეს ოპერატორთან, შემდეგ მოლარესთან და მოლარესთან საქმის დასრულების შემდეგ წავიდეს ბანკიდან.
}




int main() {
	pthread_t cashier;
	pthread_create(&cashier, NULL, Cashier, NULL);

	int i=0;
	for (; i<OPERATOR_NUM; i++) {
		pthread_t t;
		pthread_create(&t, NULL, Operator, NULL);
	}
	//ქმნის ახალ ნაკადებს შემთხვევითი დროის შუალედებით და უშვებს Client მეთოდის გამოძახებას.
	RunClientTests();
	pthread_exit(0);
	return 0;
}
```

ამოცანა 57.(2015 წლის გადაბარება) შეწებება (40 ქულა)
მოცემული გაქვთ ჯენერიკ მასივი. თქვენი ამოცანაა მასივის ელემენტები ერთმანეთს შეაწებოთ და დააბრუნოთ მიღებული შედეგის მიმთითებელი.
ორი ელემენტის შეწებებისთვის გადმოგეცემათ merge ფუნქცია
void * merge(void *elem1, void *elem2)
ჩათვალეთ რომ merge ფუნქცია უკვე დაწერილია და მას 2 არგუმენტი გადაეცემა - მასივის ელემენტებზე მიმთითებელი. ფუნქცია ელემენტებს ერთმანეთს “აწებებს” და მიღებული შედეგის მიმთითებელს აბრუნებს(ჩათვალეთ რომ შედეგიც იგივე ტიპისაა რაც მასივის ელემენტები). ანუ merge ფუნქცია ორი ელემენტის შეწეპების შედეგს აბრუნებს. ჩათვალეთ რომ იგი თვითონ გამოჰყოფს მეხსიერებას შედეგისთვის.

ა) დაწერეთ ფუნქცია merge_all რომელსაც გადაეცემა ჯენერიკ მასივი და merge ფუნქცია. merge_all ფუნქციამ უნდა იმოქმედოს შემდეგნაირად აიღოს მასივის პირველი და მეორე ელემენტი “შეაწებოს”, მიღებული შედეგი “შეაწებოს” მესამე ელემენტთან და ა.შ. სანამ ყველა ელემენტს ერთმანეთს არ “შეაწებებს” და საბოლოოდ დააბრუნოს მიღებული შედეგი.

```
void * merge_all(void * arr, int elemsize, int n, void * (merge) (const void *, const void *)){
```

ბ) დაწერეთ ფუნქცია sum_all რომელსაც გადაეცემა მთელი რიცხვების მასივი და რომელიც აბრუნებს ამ მასივში შემავალი რიცხვების ჯამს. აუცილებელია რომ

აჯამვისთვის გამოიყენოთ merge_all ფუნქცია.არ დაგავიწყდეთ შესაბამისი merge ფუნქციის იმპლემენტაცია.
int sum_all(int[] arr, int n){

გ) დაწერეთ ფუნქცია concat_all რომელსაც გადაეცემა C-სტრინგების მასივი და რომელიც აბრუნებს ერთ დიდ სტრინგს - ამ მასივში შემავალი სტრინგების გაერთიანებას. აუცილებელია გამოიყენოთ merge_all ფუნქცია. არ დაგავიწყდეთ შესაბამისი merge ფუნქციის იმპლემენტაცია.
char * concat_all(char* [] arr, int n){

აუცილებელია რომ კოდი დაიწეროს C-ზე.

ამოცანა 58.(2015 წლის გადაბარება) ანგარიშები(40 ქულა)
განათლების სამინისტროს გახსნილი აქვს N ცალი ანგარიში, რომლებზეც მუდმივად ხდება თანხის ჩარიცხვა. პარალელურად იგივე ანგარიშიდან ხდება თანხის გადარიცხვები სხვადასხვა სკოლებში და უნივერსიტეტებში. გადარიცხვისას თუ ანგარიშზე არ არის საკმარისი თანხა, მაშინ ხდება ამ თანხის მოძიება სხვა დანარჩენ ანგარიშებზე. იმ შემთხვევაში თუ მოიძებნა რამდენიმე ანგარიში რომელიც ჯამუარდ ავსებს საჭირო თანხას, გადარიცხვა სრულდება და შესაბამისი თანხა იჭრება თითოეული ანგარიშიდან. წინააღმდეგ შემთხვევაში ოპერაცია გადადის მოლოდინის რეჟიმში, მანამ სანამ არ მოხდება რაიმე ახალი ჩარიცხვა.
ახალი ჩარიცხვის შემთხვევაში თითოეული მოლოდინში მყოფი ოპერაციისთვის ხდება ხელახალი მცდელობა, თუ რომელიმეს შესრულება შესაძლებელია სრულდება, თუ არა ოპერაცია ისევ უბრუნდება მოლოდინის რეჟიმს.

AmountIn - მეთოდის გამოძახება ხდება სხვადასხვა ნაკადებიდან, როცა შემოდის მოთხოვნა თანხის ჩარიცხვაზე.
AmountOut - მეთოდის გამოძახება ხდება სხვადასხვა ნაკადებიდან, როცა შემოდის მოთხოვნა თანხის გადარიცხვაზე.
დაწერეთ ამ ორი მეთოდის რეალიზაცია ისე რომ აკმაყოფილებდეს ზემოთ ჩამოთვლილ მოთხოვნებს. შეგიძლიათ შემოიტანოთ თქვენთვის საჭირო გლობალური ცვლადები.

```
#include "helper.c"
#define N 10;
typedef struct {
        int id;
        int amount;

} Account;

Account  accounts[N];


void AmountIn(Account * fromAccount, Account * myAccount) {

}


void AmountOut(Account * myAccount, Account * toAccount) {

}
```

```
int main() {
	InitAccounts(accounts);


	pthread_exit(0);
	return 0;
}
```