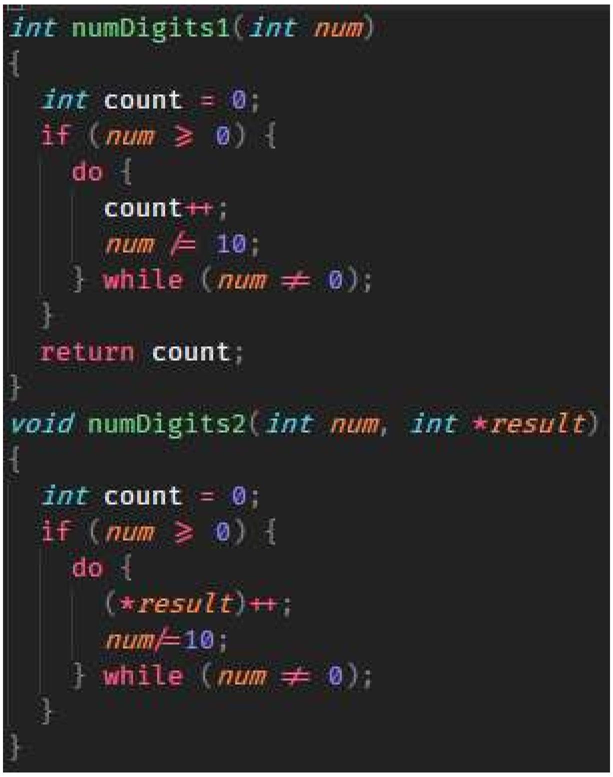
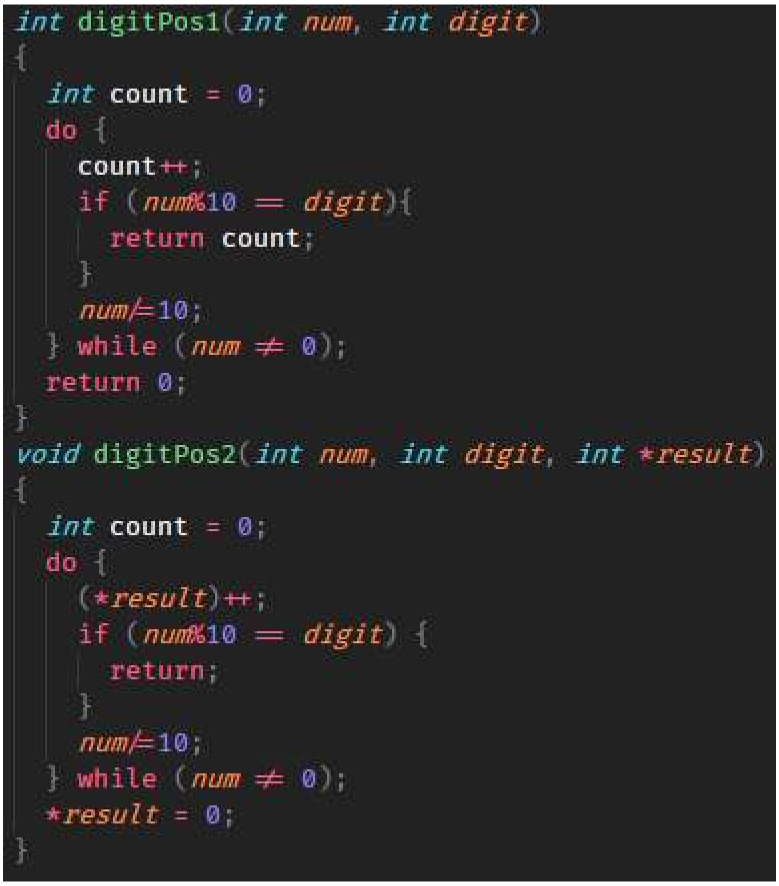
********3********

L1 numDigits



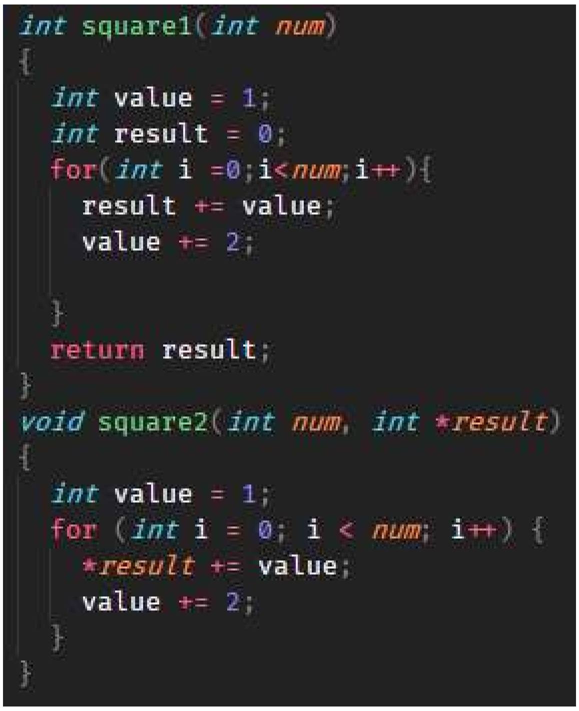
```
int numDigits1(int num)
{
  int count = 0;
  if (num >= 0) {
    do {
      count++;
      num /= 10;
    } while (num != 0);
  }
  return count;
}
void numDigits2(int num, int *result)
{
  int count = 0;
  if (num >= 0) {
    do {
      (*result)++;
      num/=10;
    } while (num != 0);
  }
}
```

L2 digitPos



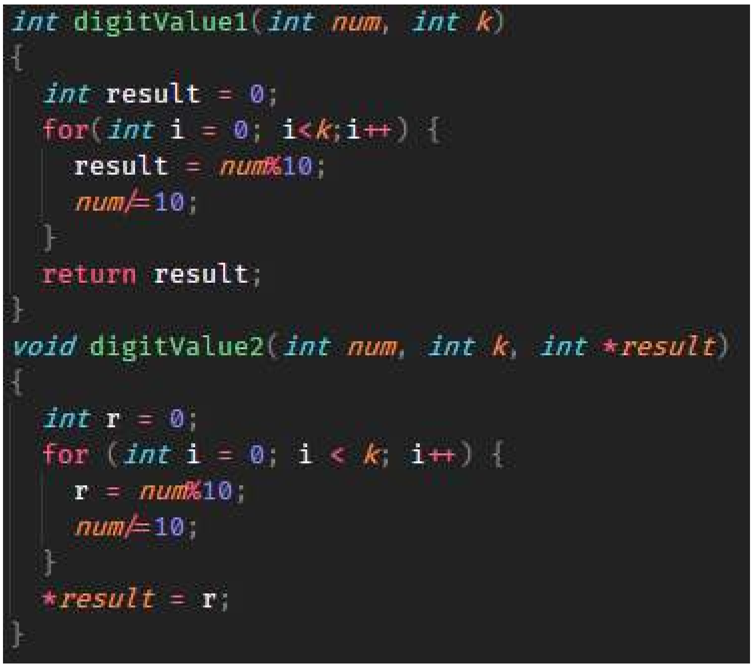
```
int digitPos1(int num, int digit)
{
  int count = 0;
  do {
    count++;
    if (num%10 == digit){
      return count;
    }
    num/=10;
  } while (num != 0);
  return 0;
}
void digitPos2(int num, int digit, int *result)
{
  int count = 0;
  do {
    (*result)++;
    if (num%10 == digit) {
      return;
    }
    num/=10;
  } while (num != 0);
  *result = 0;
}
```

L3 square



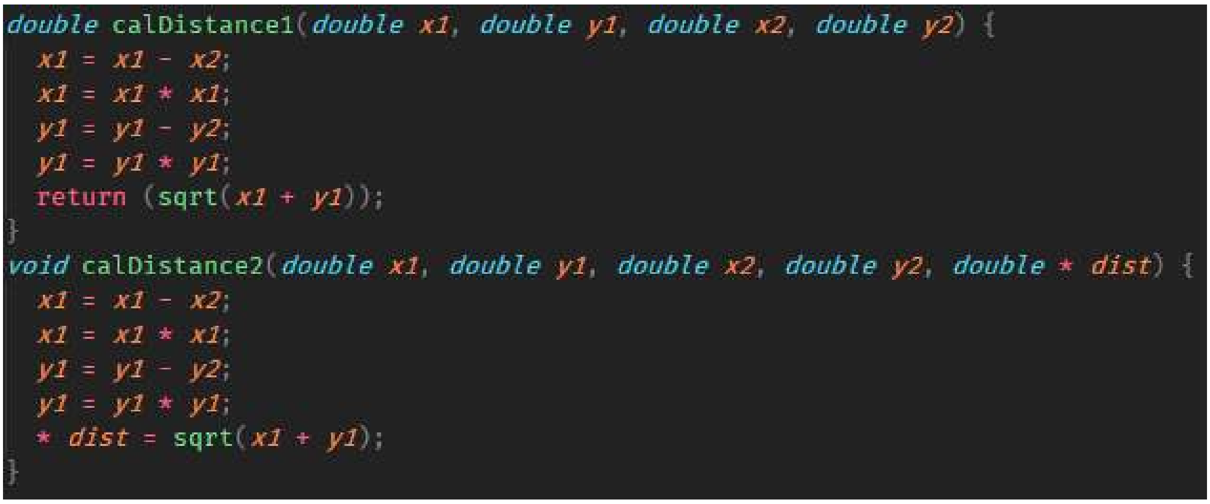
```
int square1(int num)
{
  int value = 1;
  int result = 0;
  for(int i =0;i<num;i++){
    result += value;
    value += 2;

  }
  return result;
}
void square2(int num, int *result)
{
  int value = 1;
  for (int i = 0; i < num; i++) {
    *result += value;
    value += 2;
  }
}
```

T1 digitValue



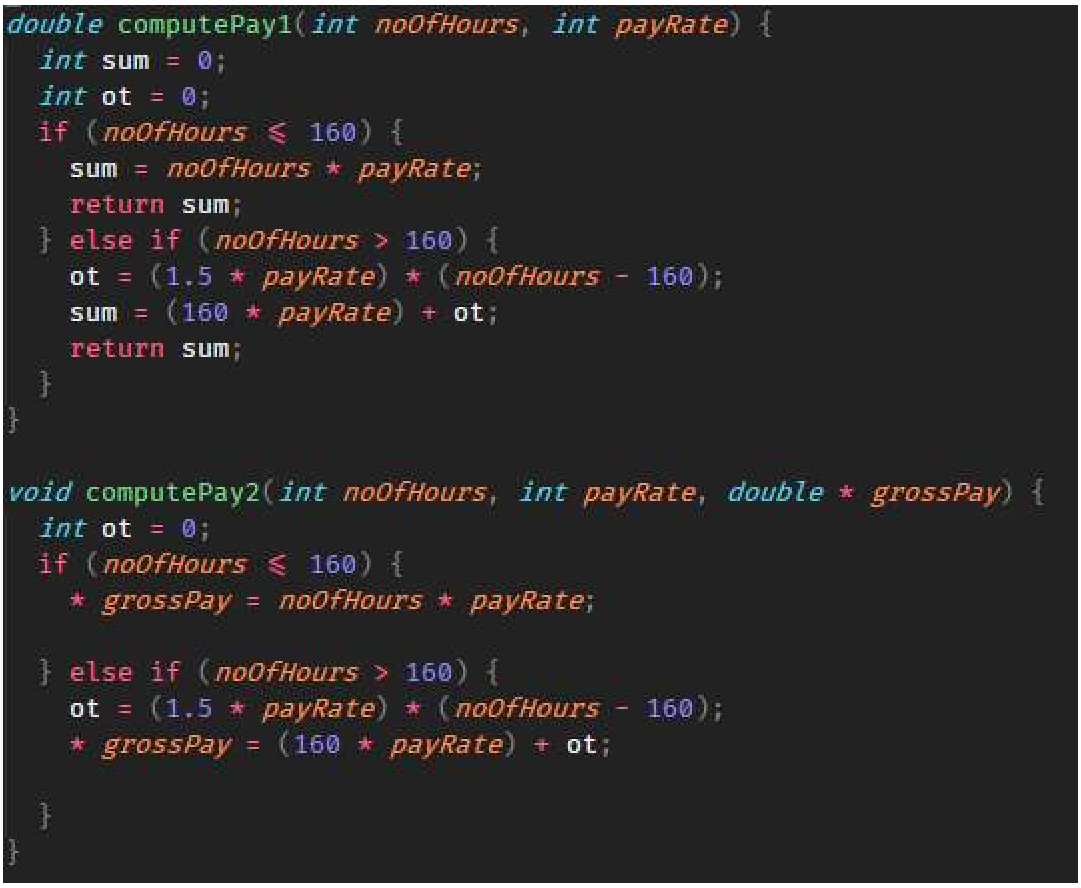
```
int digitValue1(int num, int k)
{
  int result = 0;
  for(int i = 0; i<k;i++) {
    result = num%10;
    num/=10;
  }
  return result;
}
void digitValue2(int num, int k, int *result)
{
  int r = 0;
  for (int i = 0; i < k; i++) {
    r = num%10;
    num/=10;
  }
  *result = r;
}
```

T2 calDistance



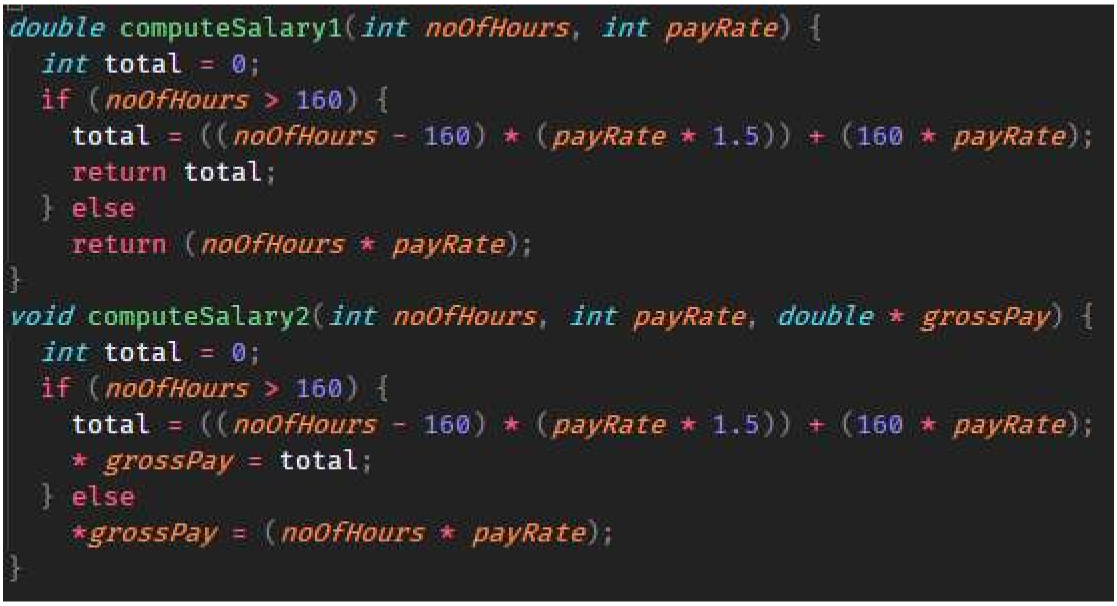
```
double calDistance1(double x1, double y1, double x2, double y2) {
  x1 = x1 - x2;
  x1 = x1 * x1;
  y1 = y1 - y2;
  y1 = y1 * y1;
  return (sqrt(x1 + y1));
}
void calDistance2(double x1, double y1, double x2, double y2, double * dist) {
  x1 = x1 - x2;
  x1 = x1 * x1;
  y1 = y1 - y2;
  y1 = y1 * y1;
  * dist = sqrt(x1 + y1);
}
```

1.computePay



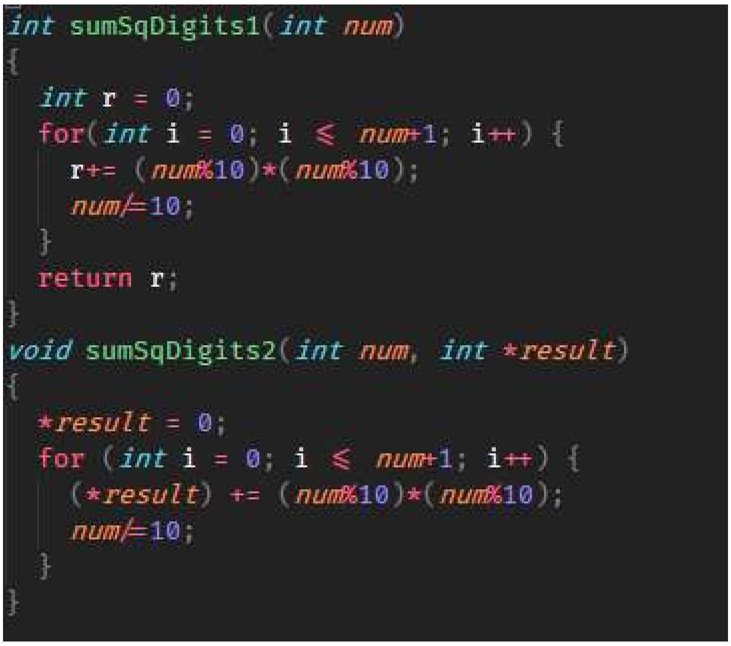
```
double computePay1(int noOfHours, int payRate) {
  int sum = 0;
  int ot = 0;
  if (noOfHours ⩽ 160) {
    sum = noOfHours * payRate;
    return sum;
  } else if (noOfHours > 160) {
    ot = (1.5 * payRate) * (noOfHours - 160);
    sum = (160 * payRate) + ot;
    return sum;
  }
}

void computePay2(int noOfHours, int payRate, double * grossPay) {
  int ot = 0;
  if (noOfHours ⩽ 160) {
    * grossPay = noOfHours * payRate;

  } else if (noOfHours > 160) {
    ot = (1.5 * payRate) * (noOfHours - 160);
    * grossPay = (160 * payRate) + ot;

  }
}
```

2.computeSalary



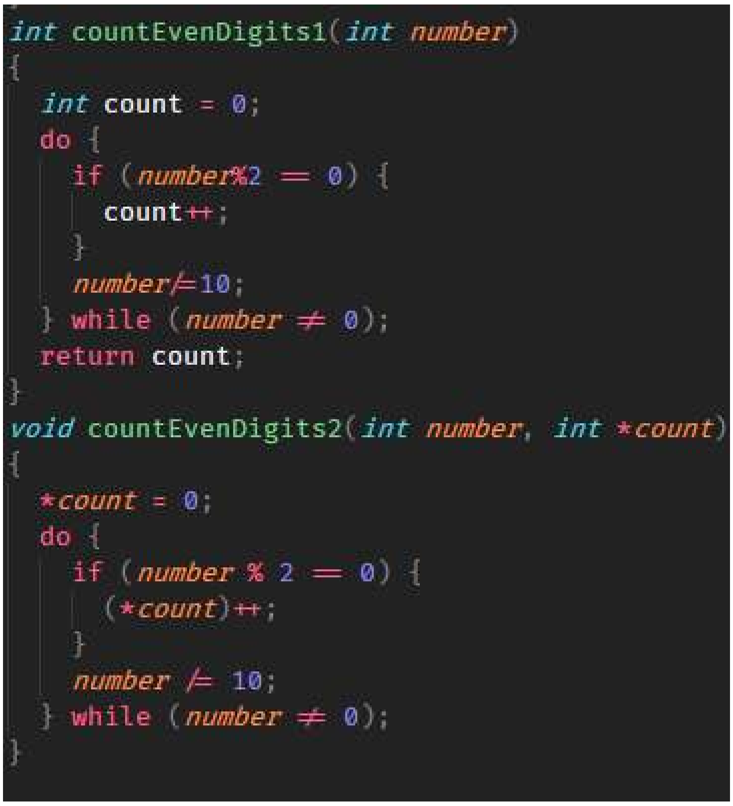
```
double computeSalary1(int noOfHours, int payRate) {
  int total = 0;
  if (noOfHours > 160) {
    total = ((noOfHours - 160) * (payRate * 1.5)) + (160 * payRate);
    return total;
  } else
    return (noOfHours * payRate);
}
void computeSalary2(int noOfHours, int payRate, double * grossPay) {
  int total = 0;
  if (noOfHours > 160) {
    total = ((noOfHours - 160) * (payRate * 1.5)) + (160 * payRate);
    * grossPay = total;
  } else
    *grossPay = (noOfHours * payRate);
}
```

3.sumSqDigits



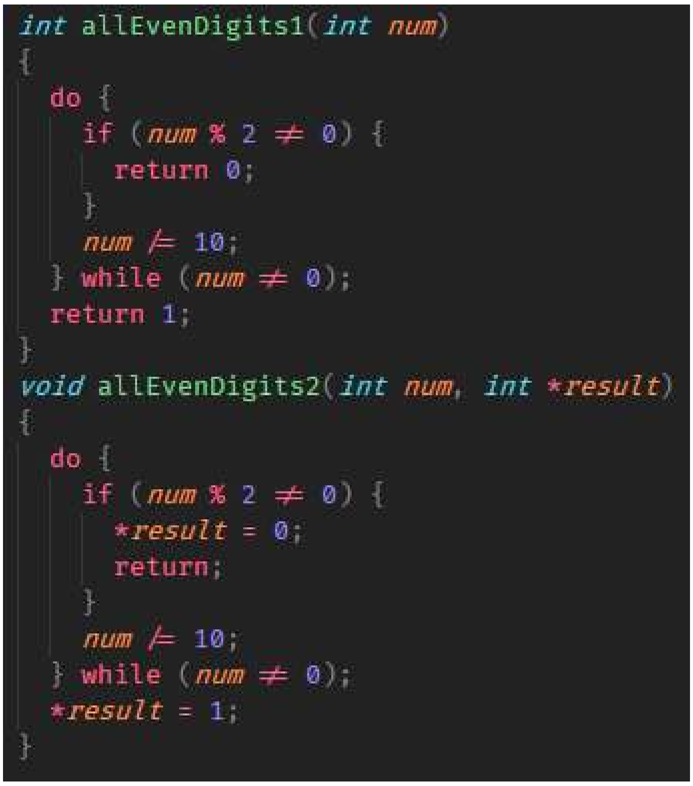
```
int sumSqDigits1(int num)
{
  int r = 0;
  for(int i = 0; i <= num+1; i++) {
    r+= (num%10)*(num%10);
    num/=10;
  }
  return r;
}
void sumSqDigits2(int num, int *result)
{
  *result = 0;
  for (int i = 0; i <= num+1; i++) {
    (*result) += (num%10)*(num%10);
    num/=10;
  }
}
```

4.countEvenDigits



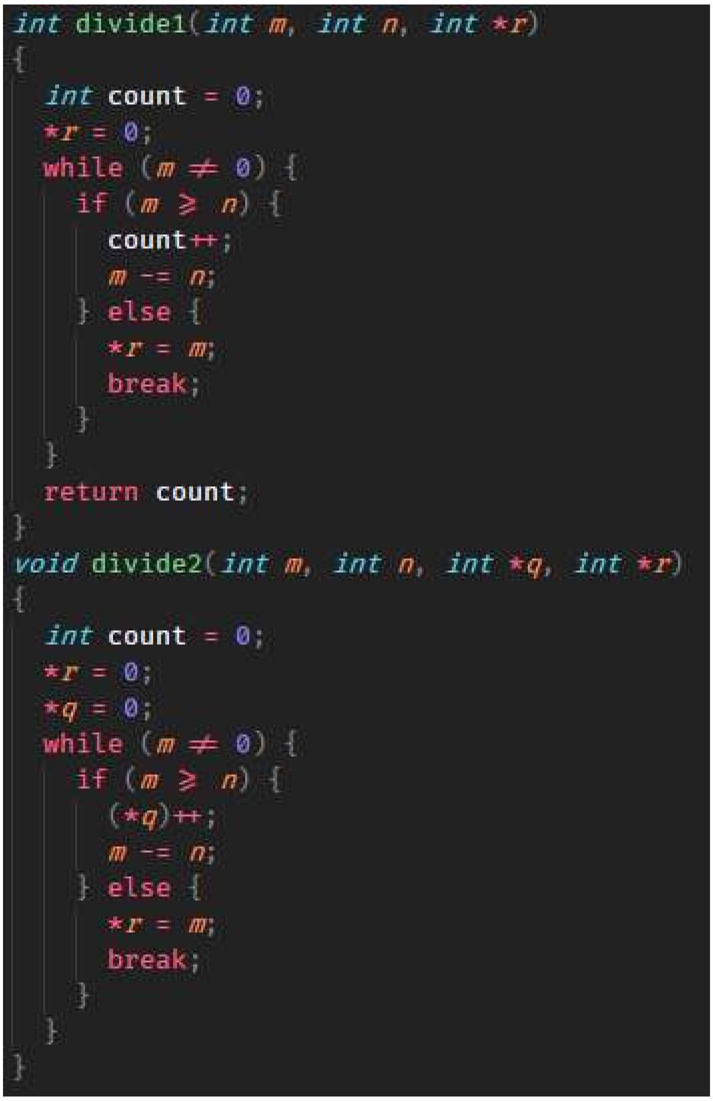
```c
int countEvenDigits1(int number)
{
  int count = 0;
  do {
    if (number%2 == 0) {
      count++;
    }
    number/=10;
  } while (number != 0);
  return count;
}
void countEvenDigits2(int number, int *count)
{
  *count = 0;
  do {
    if (number % 2 == 0) {
      (*count)++;
    }
    number /= 10;
  } while (number != 0);
}
```

5.allEvenDigits



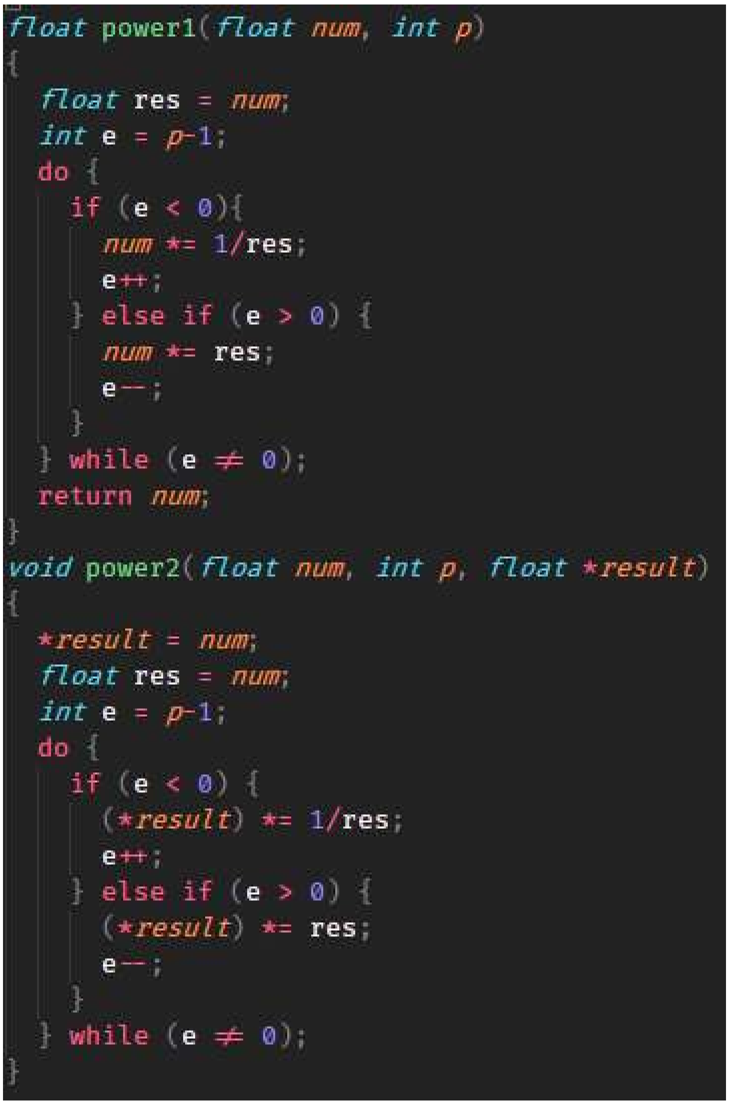
```c
int allEvenDigits1(int num)
{
  do {
    if (num % 2 != 0) {
      return 0;
    }
    num /= 10;
  } while (num != 0);
  return 1;
}
void allEvenDigits2(int num, int *result)
{
  do {
    if (num % 2 != 0) {
      *result = 0;
      return;
    }
    num /= 10;
  } while (num != 0);
  *result = 1;
}
```

6.divide



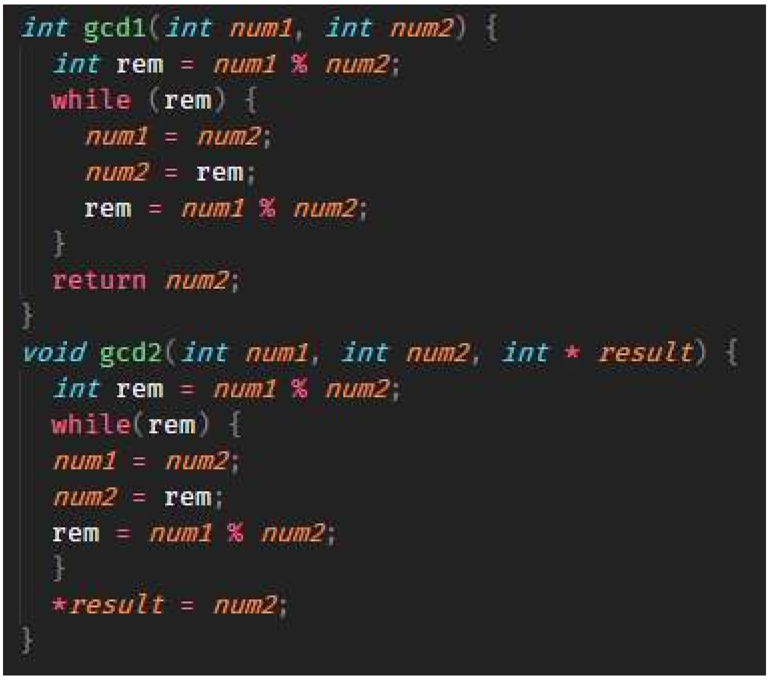
```
int divide1(int m, int n, int *r)
{
  int count = 0;
  *r = 0;
  while (m ≠ 0) {
    if (m ⩾ n) {
      count++;
      m -= n;
    } else {
      *r = m;
      break;
    }
  }
  return count;
}
void divide2(int m, int n, int *q, int *r)
{
  int count = 0;
  *r = 0;
  *q = 0;
  while (m ≠ 0) {
    if (m ⩾ n) {
      (*q)++;
      m -= n;
    } else {
      *r = m;
      break;
    }
  }
}
```

7.power



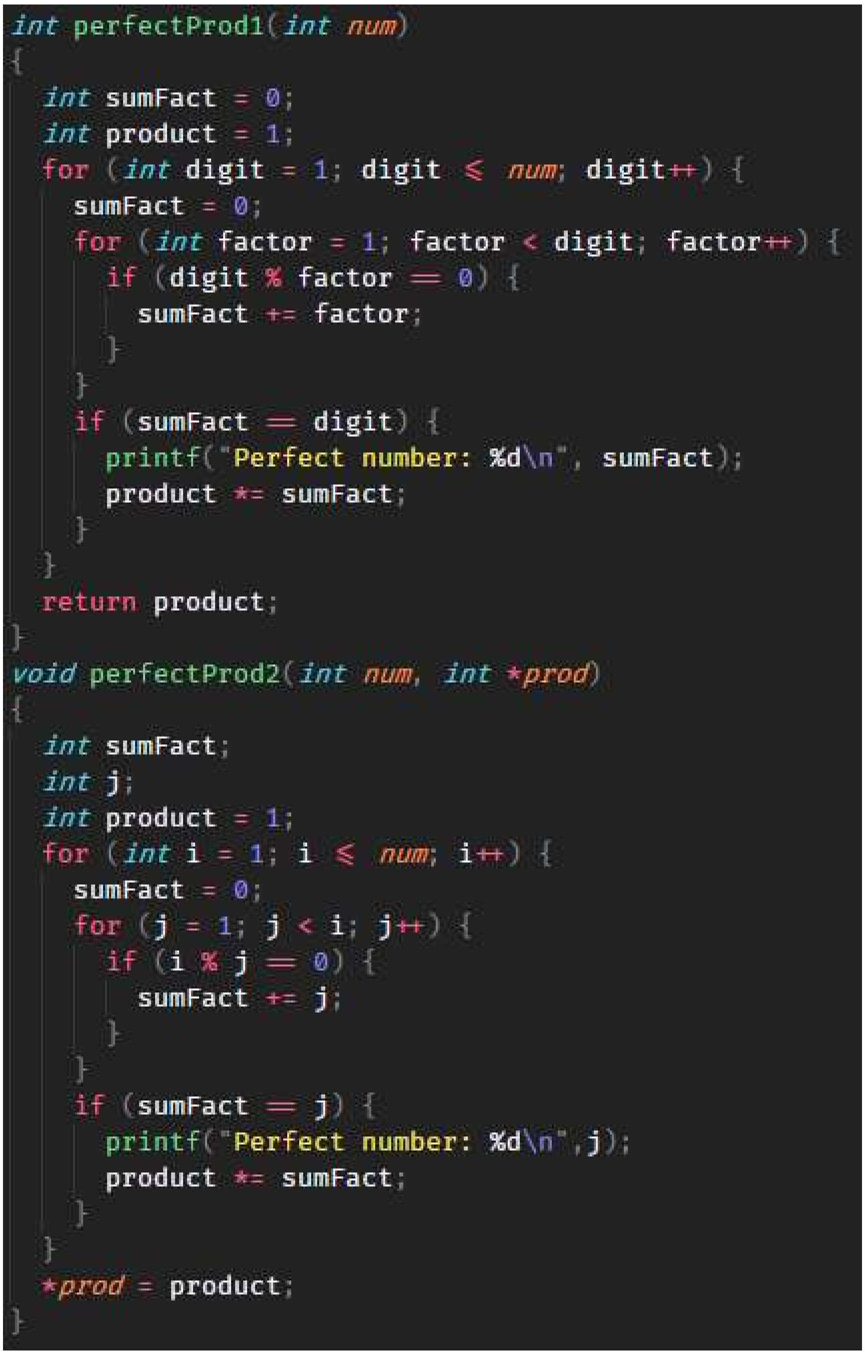
```
float power1(float num, int p)
{
  float res = num;
  int e = p-1;
  do {
    if (e < 0){
      num *= 1/res;
      e++;
    } else if (e > 0) {
      num *= res;
      e--;
    }
  } while (e ≠ 0);
  return num;
}
void power2(float num, int p, float *result)
{
  *result = num;
  float res = num;
  int e = p-1;
  do {
    if (e < 0) {
      (*result) *= 1/res;
      e++;
    } else if (e > 0) {
      (*result) *= res;
      e--;
    }
  } while (e ≠ 0);
}
```

8.gcd



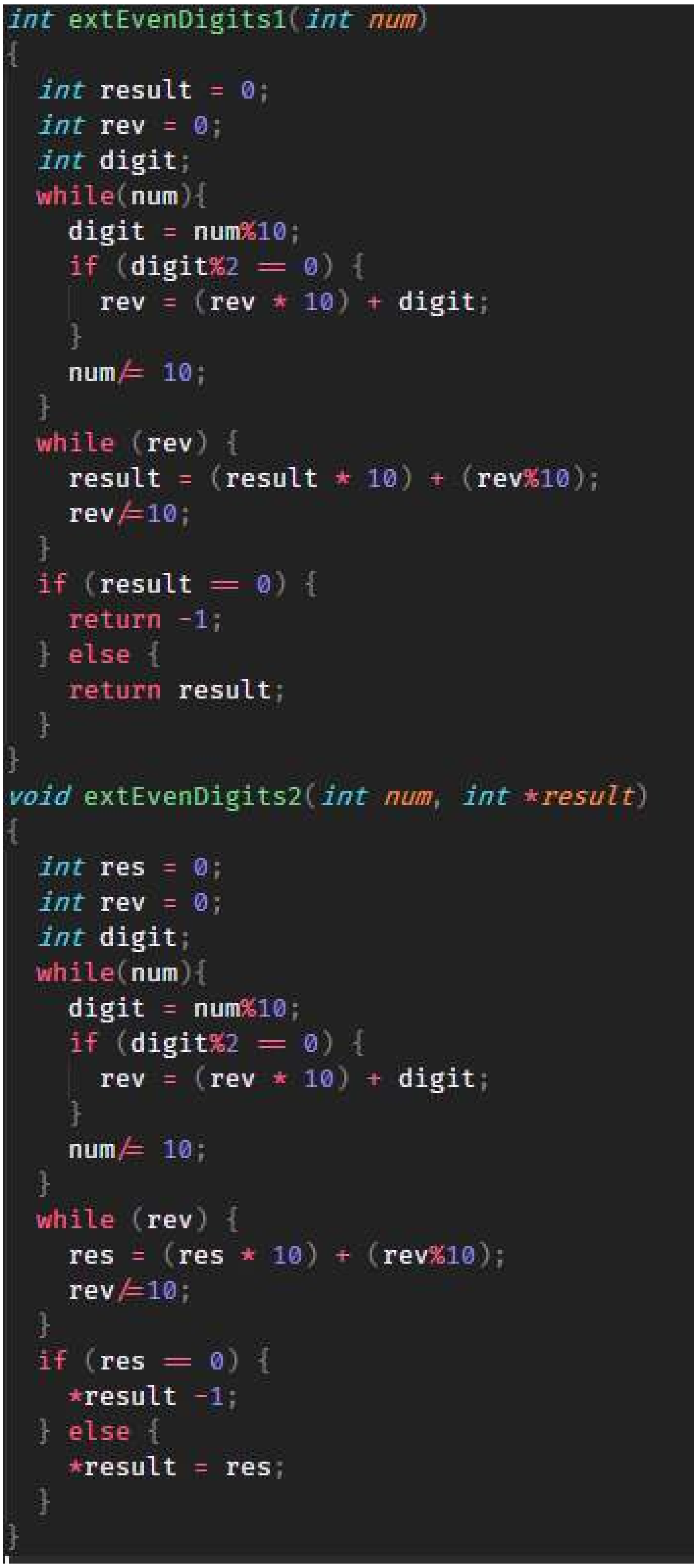
```
int gcd1(int num1, int num2) {
  int rem = num1 % num2;
  while (rem) {
    num1 = num2;
    num2 = rem;
    rem = num1 % num2;
  }
  return num2;
}
void gcd2(int num1, int num2, int * result) {
  int rem = num1 % num2;
  while(rem) {
  num1 = num2;
  num2 = rem;
  rem = num1 % num2;
  }
  *result = num2;
}
```

9.perfectProd



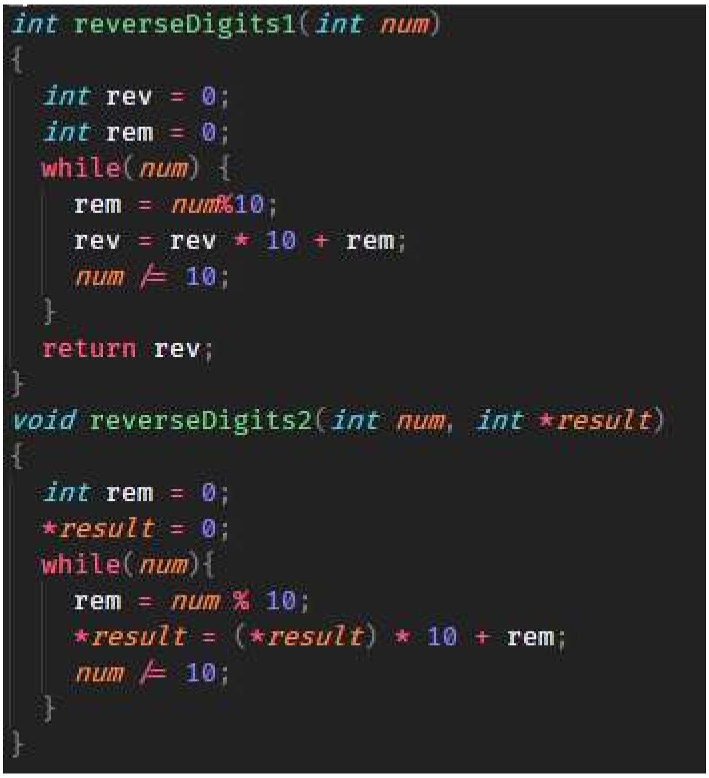
```
int perfectProd1(int num)
{
  int sumFact = 0;
  int product = 1;
  for (int digit = 1; digit <= num; digit++) {
    sumFact = 0;
    for (int factor = 1; factor < digit; factor++) {
      if (digit % factor == 0) {
        sumFact += factor;
      }
    }
    if (sumFact == digit) {
      printf("Perfect number: %d\n", sumFact);
      product *= sumFact;
    }
  }
  return product;
}
void perfectProd2(int num, int *prod)
{
  int sumFact;
  int j;
  int product = 1;
  for (int i = 1; i <= num; i++) {
    sumFact = 0;
    for (j = 1; j < i; j++) {
      if (i % j == 0) {
        sumFact += j;
      }
    }
    if (sumFact == j) {
      printf("Perfect number: %d\n",j);
      product *= sumFact;
    }
  }
  *prod = product;
}
```

10.extEvenDigits



```c
int extEvenDigits1(int num)
{
  int result = 0;
  int rev = 0;
  int digit;
  while(num){
    digit = num%10;
    if (digit%2 == 0) {
      rev = (rev * 10) + digit;
    }
    num/= 10;
  }
  while (rev) {
    result = (result * 10) + (rev%10);
    rev/=10;
  }
  if (result == 0) {
    return -1;
  } else {
    return result;
  }
}
void extEvenDigits2(int num, int *result)
{
  int res = 0;
  int rev = 0;
  int digit;
  while(num){
    digit = num%10;
    if (digit%2 == 0) {
      rev = (rev * 10) + digit;
    }
    num/= 10;
  }
  while (rev) {
    res = (res * 10) + (rev%10);
    rev/=10;
  }
  if (res == 0) {
    *result -1;
  } else {
    *result = res;
  }
}
```

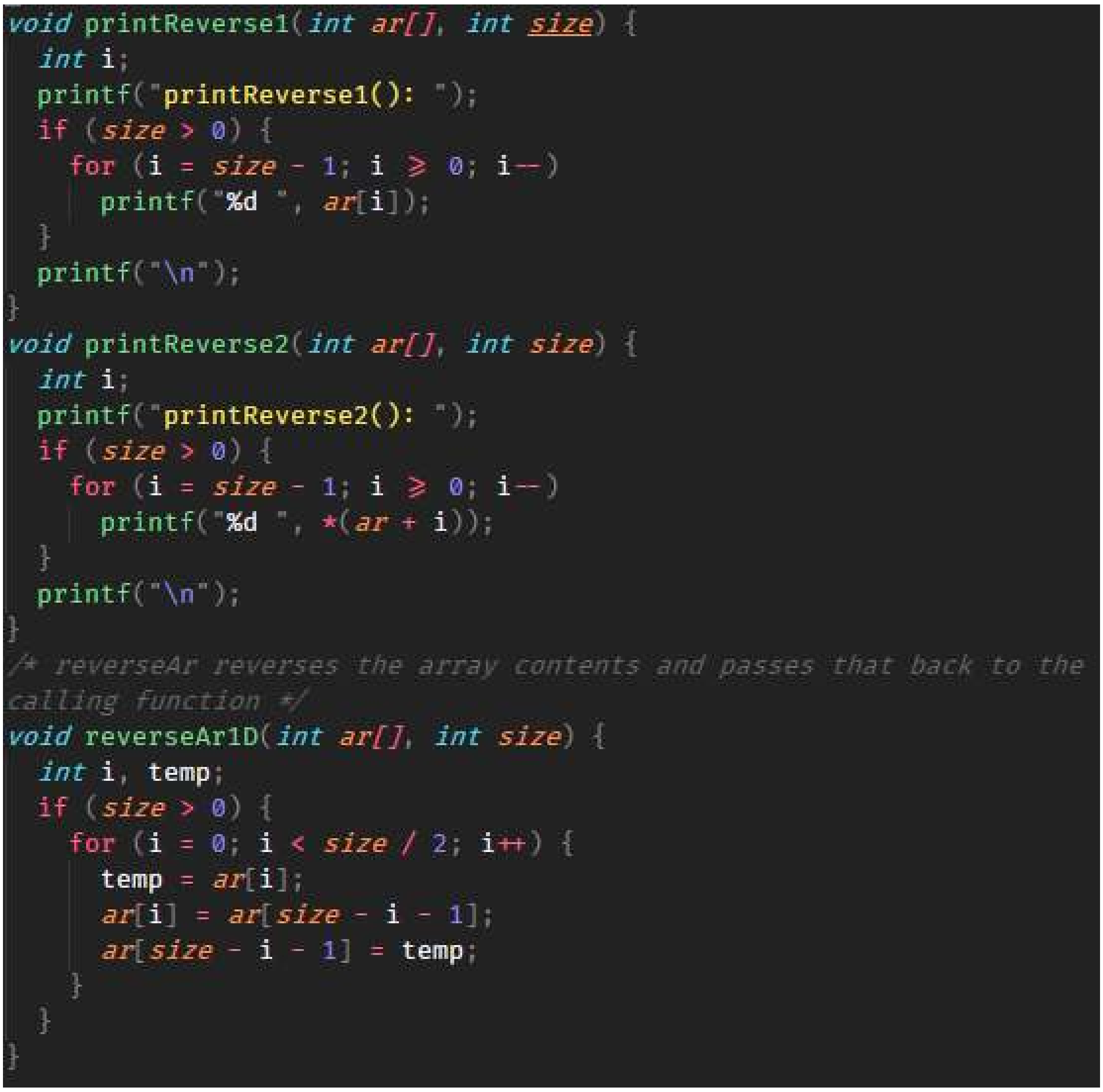
11.reverseDigits



```
int reverseDigits1(int num)
{
  int rev = 0;
  int rem = 0;
  while(num) {
    rem = num%10;
    rev = rev * 10 + rem;
    num /= 10;
  }
  return rev;
}
void reverseDigits2(int num, int *result)
{
  int rem = 0;
  *result = 0;
  while(num){
    rem = num % 10;
    *result = (*result) * 10 + rem;
    num /= 10;
  }
}
```

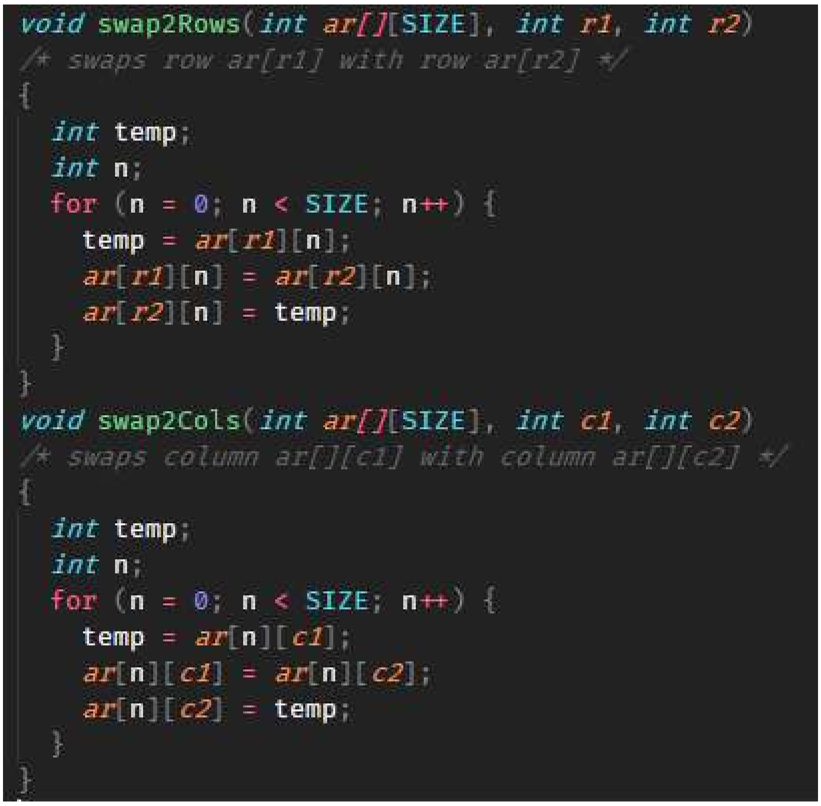
********4********

T1 reverseAr1D



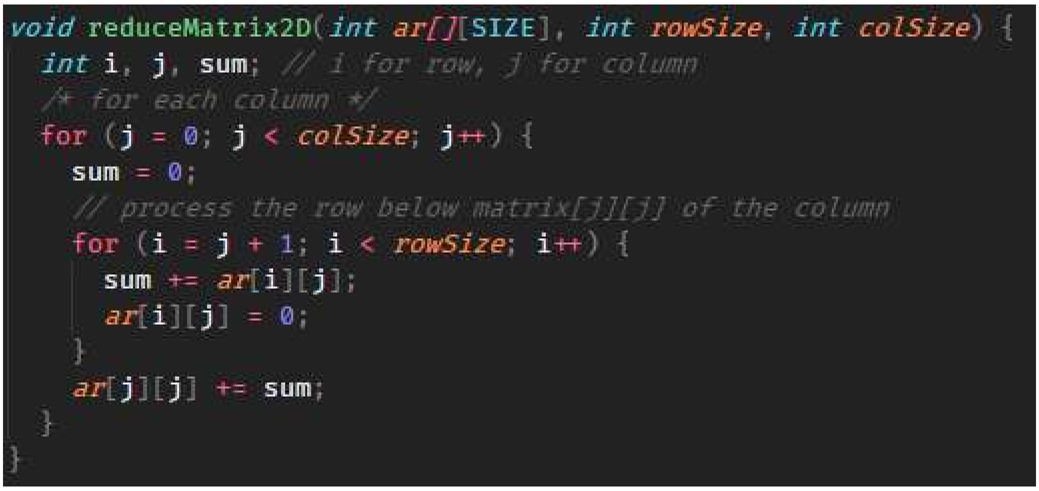
```c
void printReverse1(int ar[], int size) {
  int i;
  printf("printReverse1(): ");
  if (size > 0) {
    for (i = size - 1; i >= 0; i--)
      printf("%d ", ar[i]);
  }
  printf("\n");
}
void printReverse2(int ar[], int size) {
  int i;
  printf("printReverse2(): ");
  if (size > 0) {
    for (i = size - 1; i >= 0; i--)
      printf("%d ", *(ar + i));
  }
  printf("\n");
}
/* reverseAr reverses the array contents and passes that back to the
calling function */
void reverseAr1D(int ar[], int size) {
  int i, temp;
  if (size > 0) {
    for (i = 0; i < size / 2; i++) {
      temp = ar[i];
      ar[i] = ar[size - i - 1];
      ar[size - i - 1] = temp;
    }
  }
}
```

T2 swap2RowsCols2D



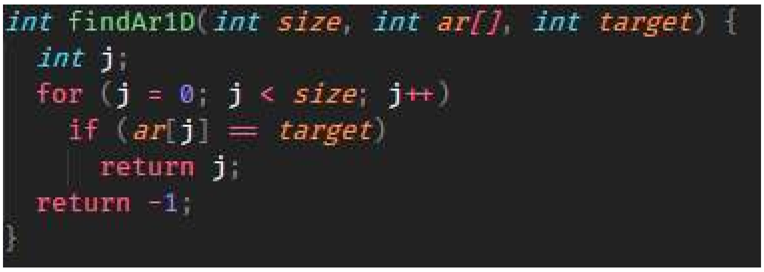
```c
void swap2Rows(int ar[][SIZE], int r1, int r2)
/* swaps row ar[r1] with row ar[r2] */
{
  int temp;
  int n;
  for (n = 0; n < SIZE; n++) {
    temp = ar[r1][n];
    ar[r1][n] = ar[r2][n];
    ar[r2][n] = temp;
  }
}
void swap2Cols(int ar[][SIZE], int c1, int c2)
/* swaps column ar[][c1] with column ar[][c2] */
{
  int temp;
  int n;
  for (n = 0; n < SIZE; n++) {
    temp = ar[n][c1];
    ar[n][c1] = ar[n][c2];
    ar[n][c2] = temp;
  }
}
```

T3 reduceMatrix2D



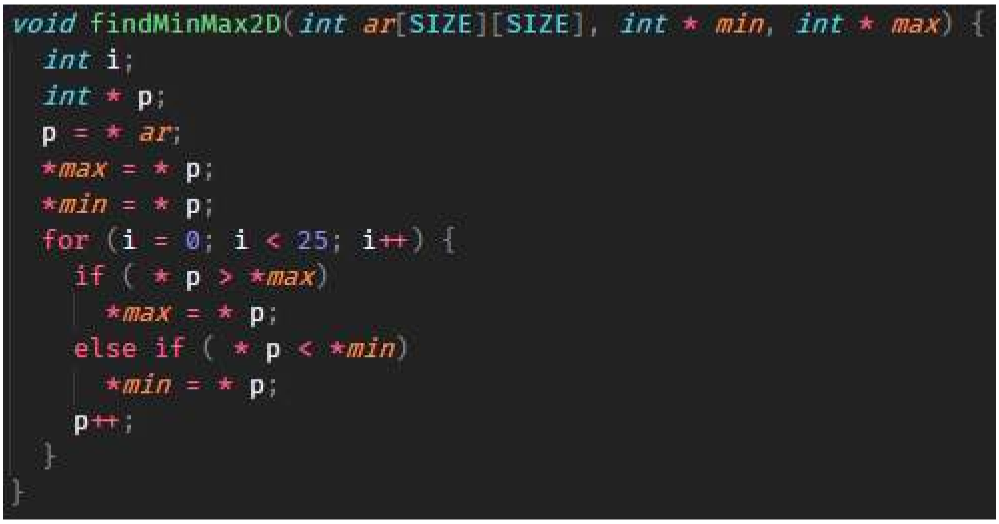
```c
void reduceMatrix2D(int ar[][SIZE], int rowSize, int colSize) {
  int i, j, sum; // i for row, j for column
  /* for each column */
  for (j = 0; j < colSize; j++) {
    sum = 0;
    // process the row below matrix[j][j] of the column
    for (i = j + 1; i < rowSize; i++) {
      sum += ar[i][j];
      ar[i][j] = 0;
    }
    ar[j][j] += sum;
  }
}
```

L1 findAr1D



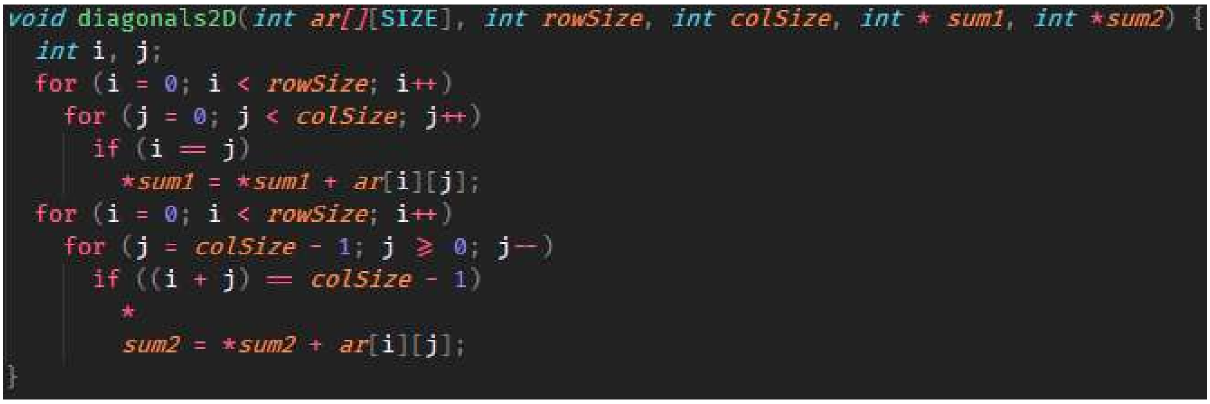
```c
int findAr1D(int size, int ar[], int target) {
  int j;
  for (j = 0; j < size; j++)
    if (ar[j] == target)
      return j;
  return -1;
}
```

L2 findMinMax2D



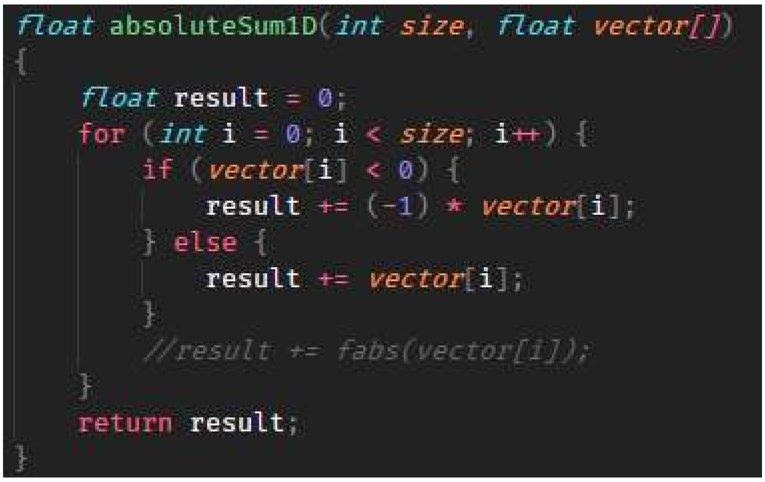
```
void findMinMax2D(int ar[SIZE][SIZE], int * min, int * max) {
  int i;
  int * p;
  p = * ar;
  *max = * p;
  *min = * p;
  for (i = 0; i < 25; i++) {
    if ( * p > *max)
      *max = * p;
    else if ( * p < *min)
      *min = * p;
    p++;
  }
}
```

L3 diagonals2D



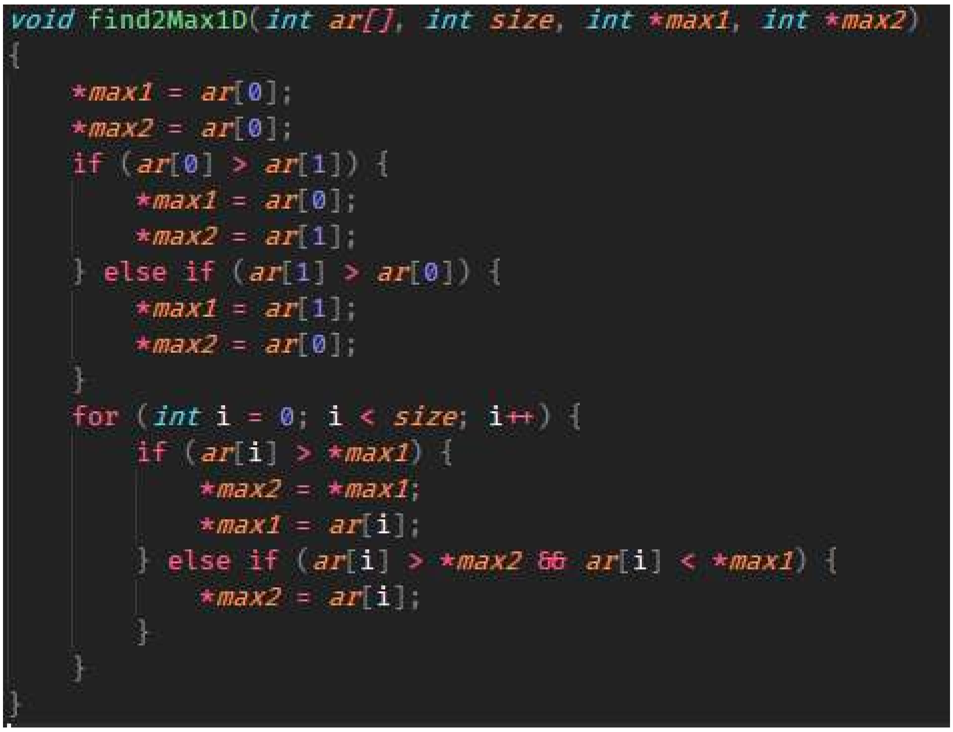
```
void diagonals2D(int ar[][SIZE], int rowSize, int colSize, int * sum1, int *sum2) {
  int i, j;
  for (i = 0; i < rowSize; i++)
    for (j = 0; j < colSize; j++)
      if (i == j)
        *sum1 = *sum1 + ar[i][j];
  for (i = 0; i < rowSize; i++)
    for (j = colSize - 1; j >= 0; j--)
      if ((i + j) == colSize - 1)
        *
        sum2 = *sum2 + ar[i][j];
}
```

1.absoluteSum1D



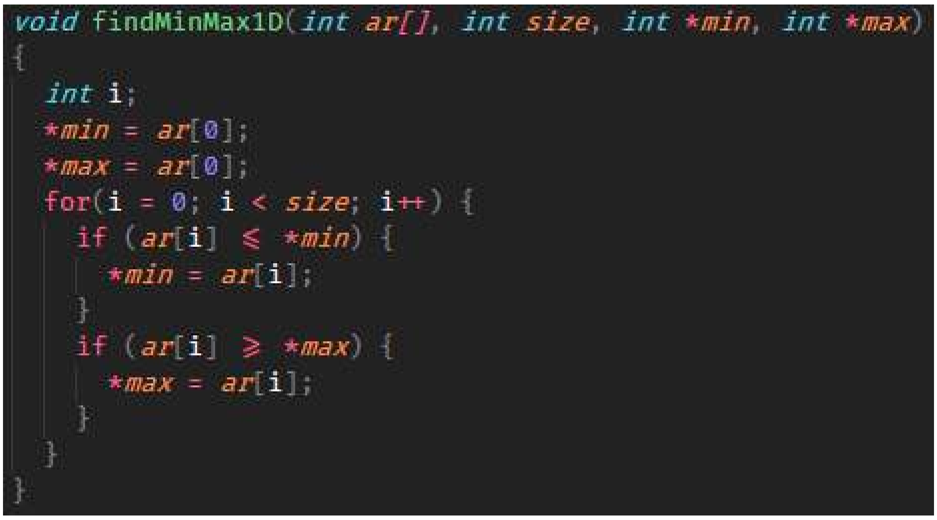
```
float absoluteSum1D(int size, float vector[])
{
    float result = 0;
    for (int i = 0; i < size; i++) {
        if (vector[i] < 0) {
            result += (-1) * vector[i];
        } else {
            result += vector[i];
        }
        //result += fabs(vector[i]);
    }
    return result;
}
```

2.find2Max1D



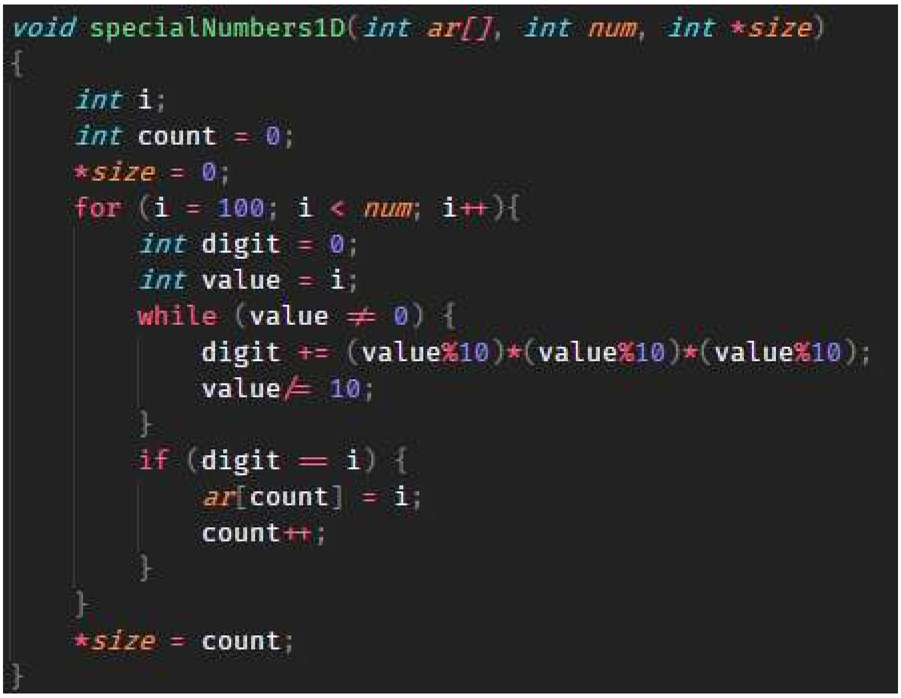
```c
void find2Max1D(int ar[], int size, int *max1, int *max2)
{
    *max1 = ar[0];
    *max2 = ar[0];
    if (ar[0] > ar[1]) {
        *max1 = ar[0];
        *max2 = ar[1];
    } else if (ar[1] > ar[0]) {
        *max1 = ar[1];
        *max2 = ar[0];
    }
    for (int i = 0; i < size; i++) {
        if (ar[i] > *max1) {
            *max2 = *max1;
            *max1 = ar[i];
        } else if (ar[i] > *max2 && ar[i] < *max1) {
            *max2 = ar[i];
        }
    }
}
```

3.findMinMax1D



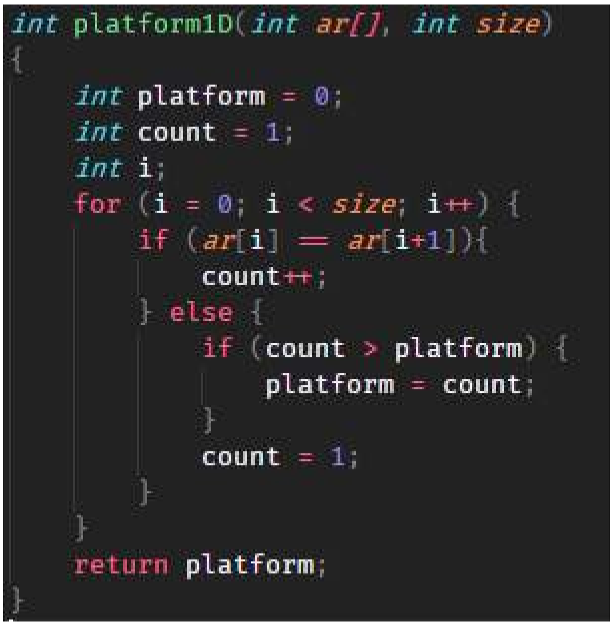
```c
void findMinMax1D(int ar[], int size, int *min, int *max)
{
  int i;
  *min = ar[0];
  *max = ar[0];
  for(i = 0; i < size; i++) {
    if (ar[i] <= *min) {
      *min = ar[i];
    }
    if (ar[i] >= *max) {
      *max = ar[i];
    }
  }
}
```

4.specialNumbers1D



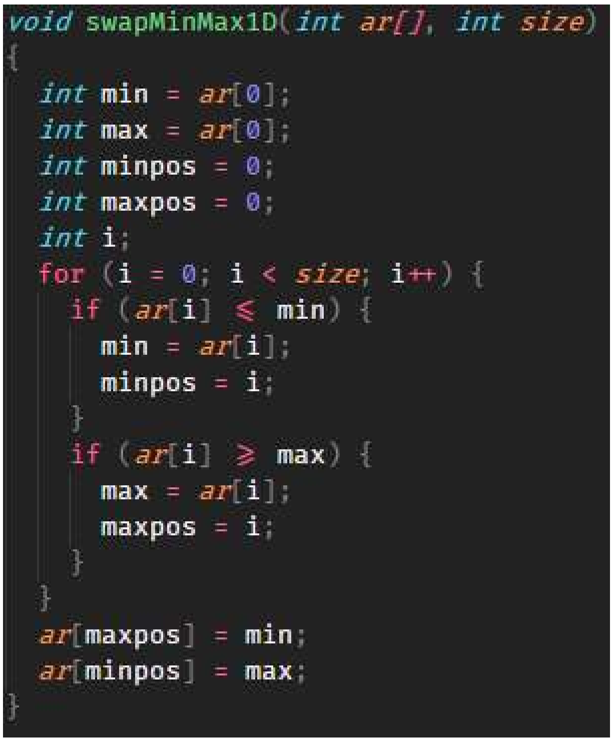
```c
void specialNumbers1D(int ar[], int num, int *size)
{
    int i;
    int count = 0;
    *size = 0;
    for (i = 100; i < num; i++){
        int digit = 0;
        int value = i;
        while (value != 0) {
            digit += (value%10)*(value%10)*(value%10);
            value/= 10;
        }
        if (digit == i) {
            ar[count] = i;
            count++;
        }
    }
    *size = count;
}
```

5.platform1D



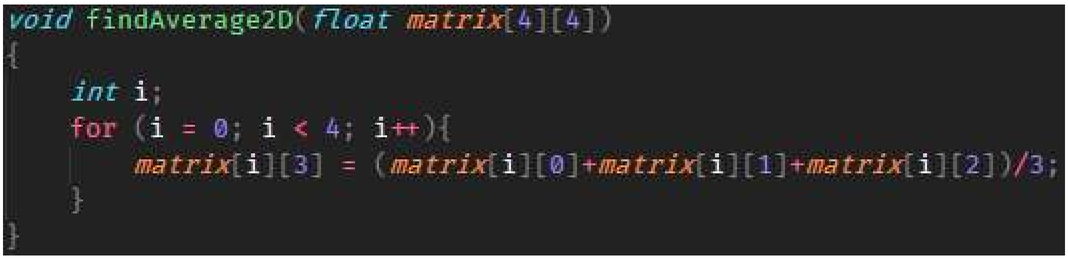
```c
int platform1D(int ar[], int size)
{
    int platform = 0;
    int count = 1;
    int i;
    for (i = 0; i < size; i++) {
        if (ar[i] == ar[i+1]){
            count++;
        } else {
            if (count > platform) {
                platform = count;
            }
            count = 1;
        }
    }
    return platform;
}
```

6.swapMinMax1D *



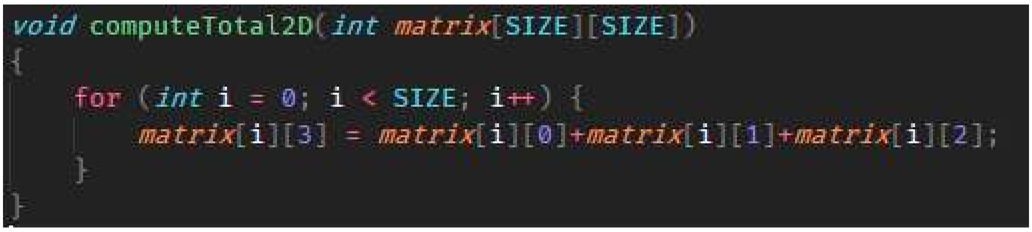
```
void swapMinMax1D(int ar[], int size)
{
  int min = ar[0];
  int max = ar[0];
  int minpos = 0;
  int maxpos = 0;
  int i;
  for (i = 0; i < size; i++) {
    if (ar[i] <= min) {
      min = ar[i];
      minpos = i;
    }
    if (ar[i] >= max) {
      max = ar[i];
      maxpos = i;
    }
  }
  ar[maxpos] = min;
  ar[minpos] = max;
}
```

7.findAverage2D



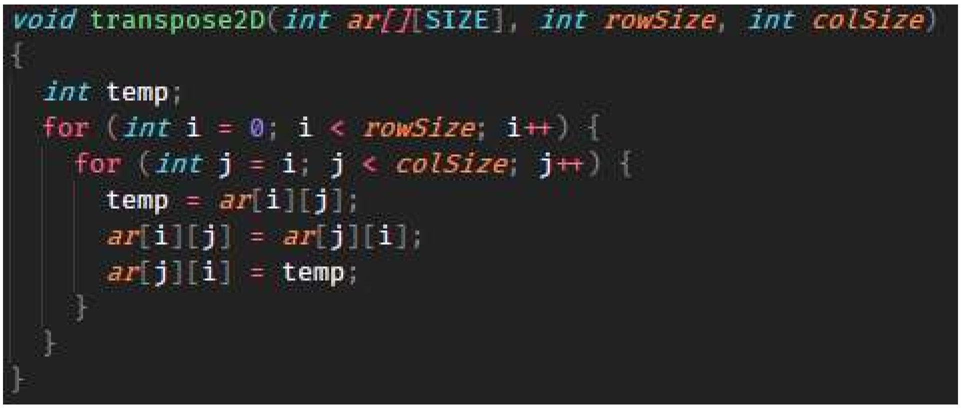
```
void findAverage2D(float matrix[4][4])
{
    int i;
    for (i = 0; i < 4; i++){
        matrix[i][3] = (matrix[i][0]+matrix[i][1]+matrix[i][2])/3;
    }
}
```

8.computeTotal2D



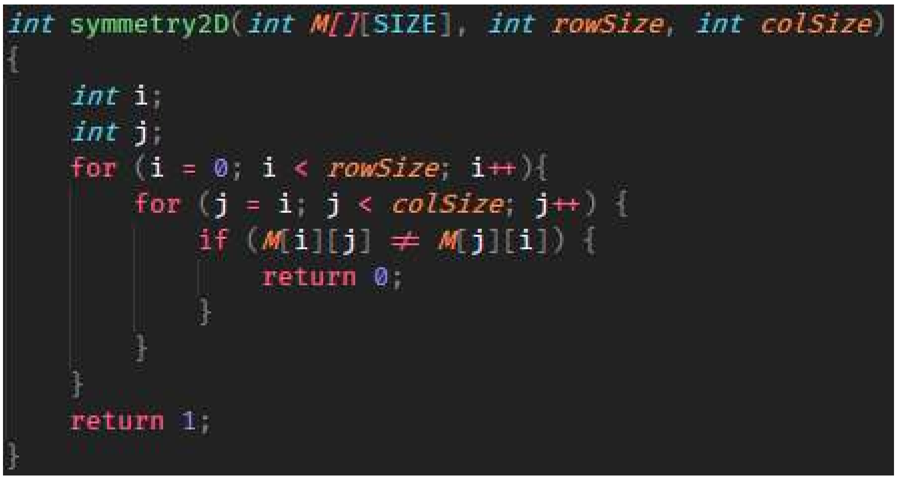
```
void computeTotal2D(int matrix[SIZE][SIZE])
{
    for (int i = 0; i < SIZE; i++) {
        matrix[i][3] = matrix[i][0]+matrix[i][1]+matrix[i][2];
    }
}
```

9.transpose2D *



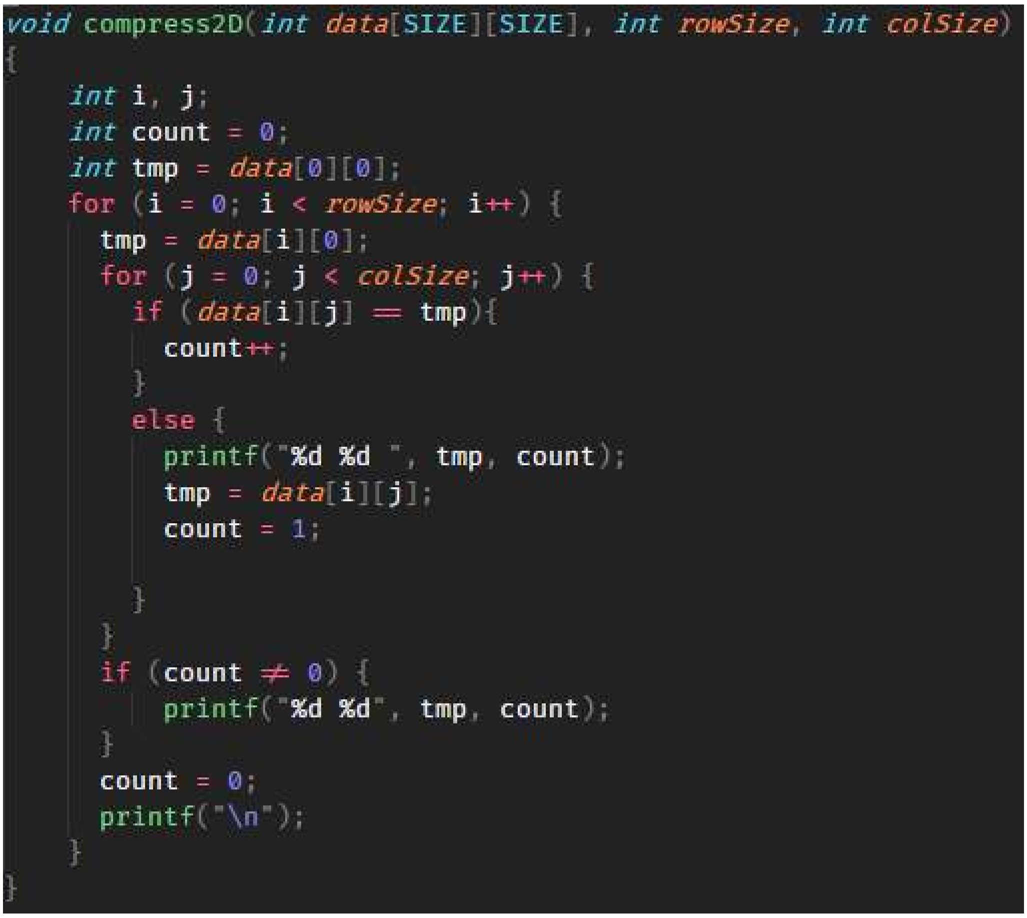
```c
void transpose2D(int ar[][SIZE], int rowSize, int colSize)
{
  int temp;
  for (int i = 0; i < rowSize; i++) {
    for (int j = i; j < colSize; j++) {
      temp = ar[i][j];
      ar[i][j] = ar[j][i];
      ar[j][i] = temp;
    }
  }
}
```

10.symmetry2D



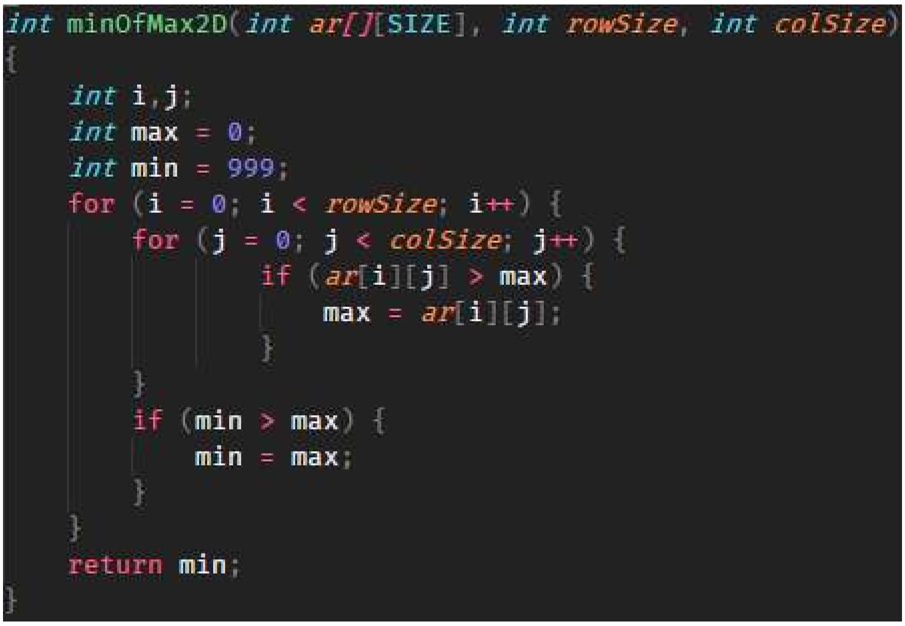
```c
int symmetry2D(int M[][SIZE], int rowSize, int colSize)
{
    int i;
    int j;
    for (i = 0; i < rowSize; i++){
        for (j = i; j < colSize; j++) {
            if (M[i][j] ≠ M[j][i]) {
                return 0;
            }
        }
    }
    return 1;
}
```

11.compress2D



```c
void compress2D(int data[SIZE][SIZE], int rowSize, int colSize)
{
    int i, j;
    int count = 0;
    int tmp = data[0][0];
    for (i = 0; i < rowSize; i++) {
      tmp = data[i][0];
      for (j = 0; j < colSize; j++) {
        if (data[i][j] == tmp){
          count++;
        }
        else {
          printf("%d %d ", tmp, count);
          tmp = data[i][j];
          count = 1;

        }
      }
      if (count != 0) {
          printf("%d %d", tmp, count);
      }
      count = 0;
      printf("\n");
    }
}
```

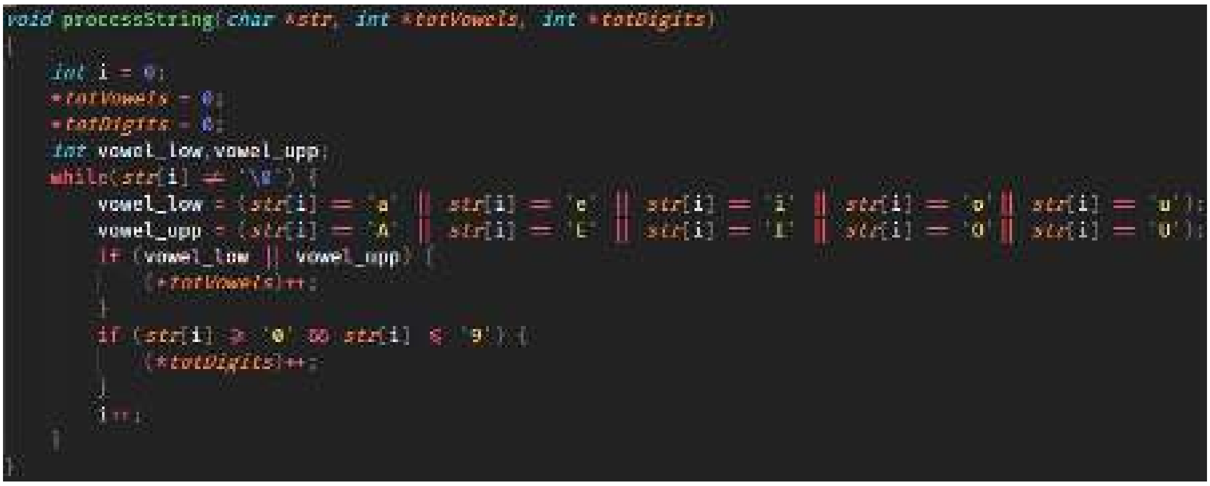
12.minOfMax2D



```c
int minOfMax2D(int ar[][SIZE], int rowSize, int colSize)
{
    int i,j;
    int max = 0;
    int min = 999;
    for (i = 0; i < rowSize; i++) {
        for (j = 0; j < colSize; j++) {
                if (ar[i][j] > max) {
                    max = ar[i][j];
                }
        }
        if (min > max) {
            min = max;
        }
    }
    return min;
}
```

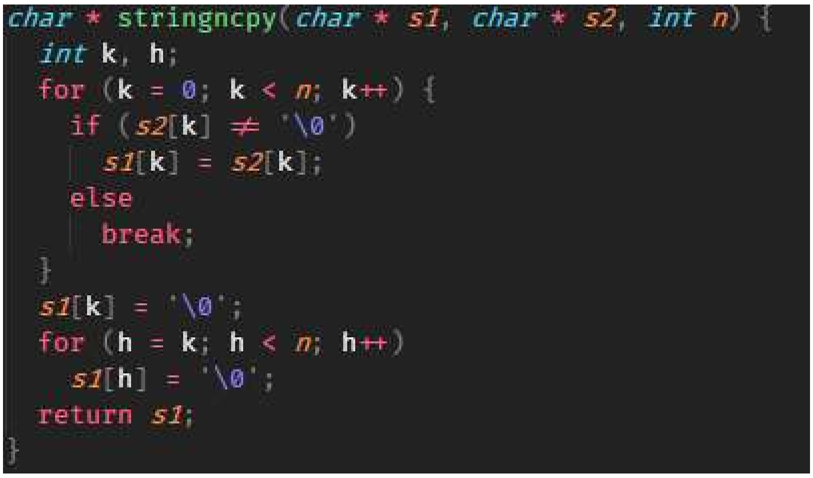
********5********

T1 processString



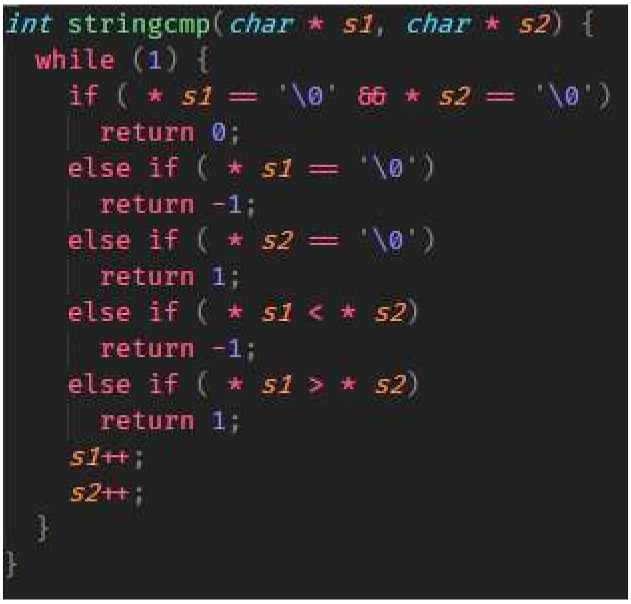
```
void processString(char *str, int *totVowels, int *totDigits)
{
    int i = 0;
    *totVowels = 0;
    *totDigits = 0;
    int vowel_low,vowel_upp;
    while(str[i] ≠ '\0') {
        vowel_low = (str[i] == 'a' || str[i] == 'e' || str[i] == 'i' || str[i] == 'o' || str[i] == 'u');
        vowel_upp = (str[i] == 'A' || str[i] == 'E' || str[i] == 'I' || str[i] == 'O' || str[i] == 'U');
        if (vowel_low || vowel_upp) {
            (*totVowels)++;
        }
        if (str[i] ≥ '0' && str[i] ≤ '9') {
            (*totDigits)++;
        }
        i++;
    }
}
```

T2 stringncpy



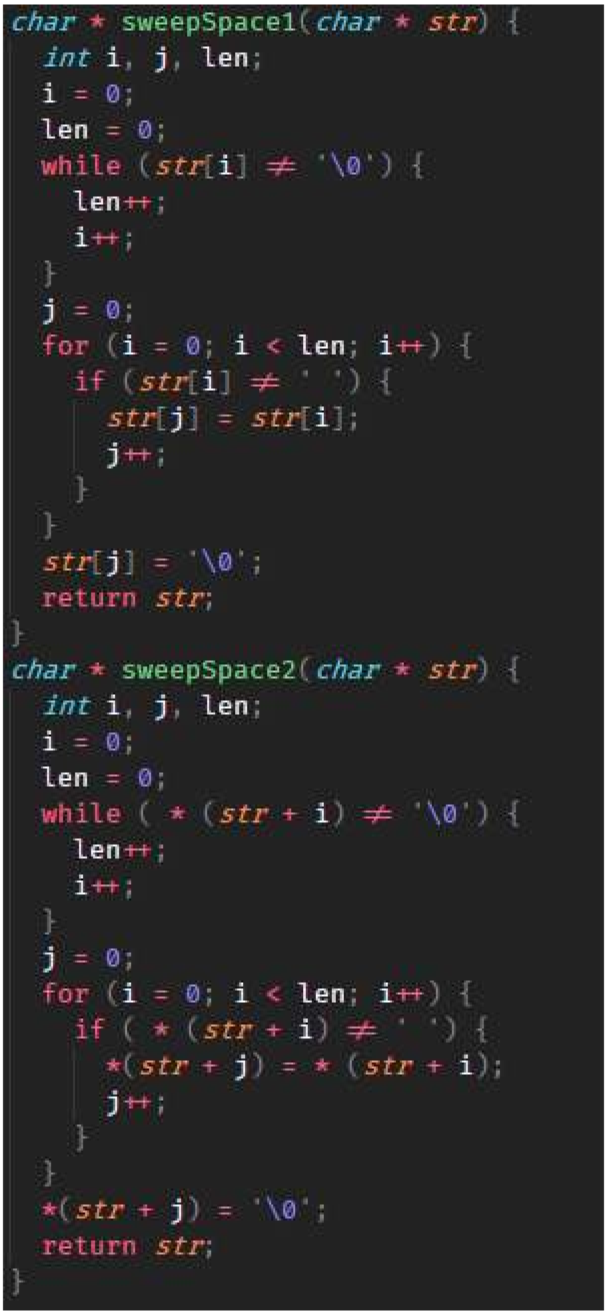
```
char * stringncpy(char * s1, char * s2, int n) {
  int k, h;
  for (k = 0; k < n; k++) {
    if (s2[k] ≠ '\0')
      s1[k] = s2[k];
    else
      break;
  }
  s1[k] = '\0';
  for (h = k; h < n; h++)
    s1[h] = '\0';
  return s1;
}
```

T3 stringcmp



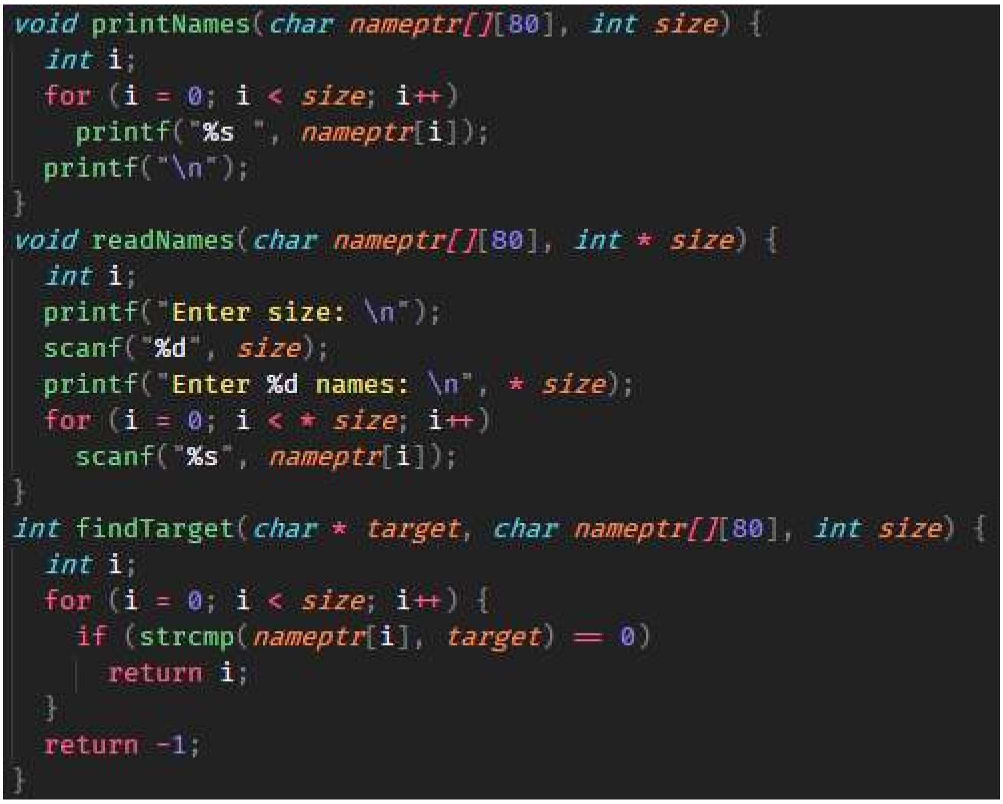
```
int stringcmp(char * s1, char * s2) {
  while (1) {
    if ( * s1 == '\0' && * s2 == '\0')
      return 0;
    else if ( * s1 == '\0')
      return -1;
    else if ( * s2 == '\0')
      return 1;
    else if ( * s1 < * s2)
      return -1;
    else if ( * s1 > * s2)
      return 1;
    s1++;
    s2++;
  }
}
```

L1 sweepSpace



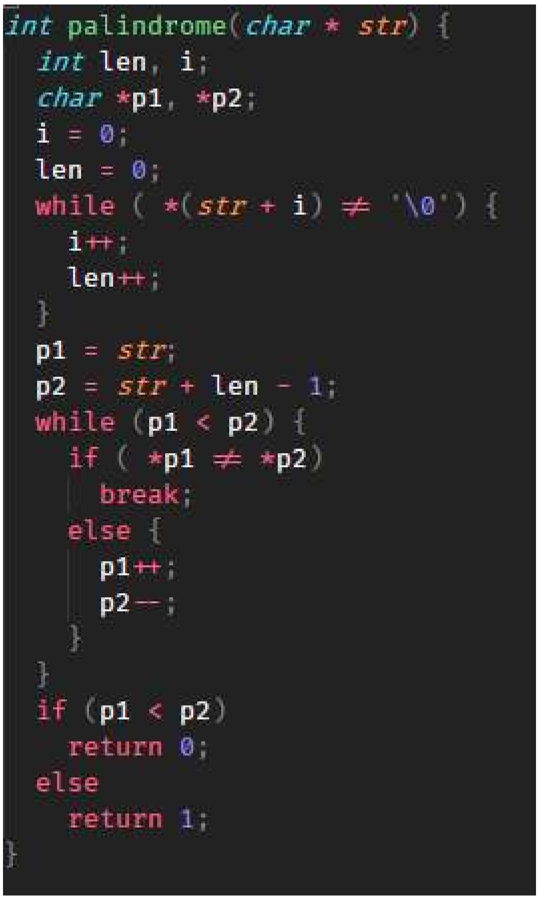
```
char * sweepSpace1(char * str) {
  int i, j, len;
  i = 0;
  len = 0;
  while (str[i] ≠ '\0') {
    len++;
    i++;
  }
  j = 0;
  for (i = 0; i < len; i++) {
    if (str[i] ≠ ' ') {
      str[j] = str[i];
      j++;
    }
  }
  str[j] = '\0';
  return str;
}
char * sweepSpace2(char * str) {
  int i, j, len;
  i = 0;
  len = 0;
  while ( * (str + i) ≠ '\0') {
    len++;
    i++;
  }
  j = 0;
  for (i = 0; i < len; i++) {
    if ( * (str + i) ≠ ' ') {
      *(str + j) = * (str + i);
      j++;
    }
  }
  *(str + j) = '\0';
  return str;
}
```

L2 findTarget



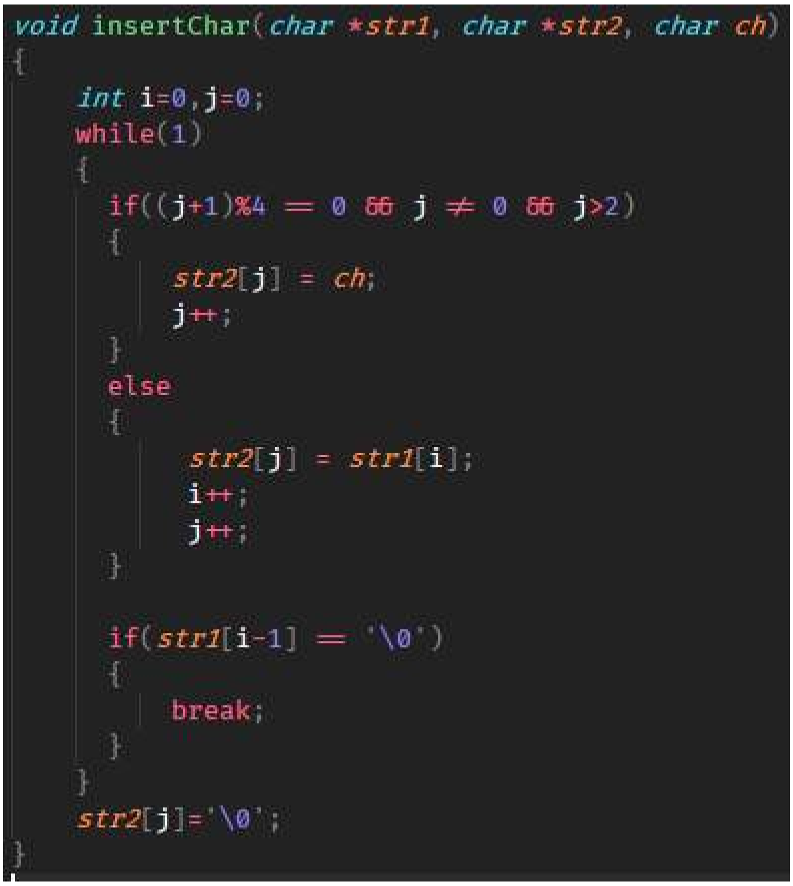
```
void printNames(char nameptr[][80], int size) {
  int i;
  for (i = 0; i < size; i++)
    printf("%s ", nameptr[i]);
  printf("\n");
}
void readNames(char nameptr[][80], int * size) {
  int i;
  printf("Enter size: \n");
  scanf("%d", size);
  printf("Enter %d names: \n", * size);
  for (i = 0; i < * size; i++)
    scanf("%s", nameptr[i]);
}
int findTarget(char * target, char nameptr[][80], int size) {
  int i;
  for (i = 0; i < size; i++) {
    if (strcmp(nameptr[i], target) == 0)
      return i;
  }
  return -1;
}
```

L3 palindrome



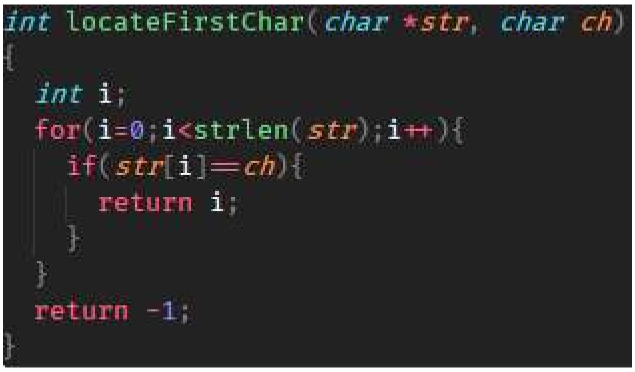
```
int palindrome(char * str) {
  int len, i;
  char *p1, *p2;
  i = 0;
  len = 0;
  while ( *(str + i) != '\0') {
    i++;
    len++;
  }
  p1 = str;
  p2 = str + len - 1;
  while (p1 < p2) {
    if ( *p1 != *p2)
      break;
    else {
      p1++;
      p2--;
    }
  }
  if (p1 < p2)
    return 0;
  else
    return 1;
}
```

1.insertChar



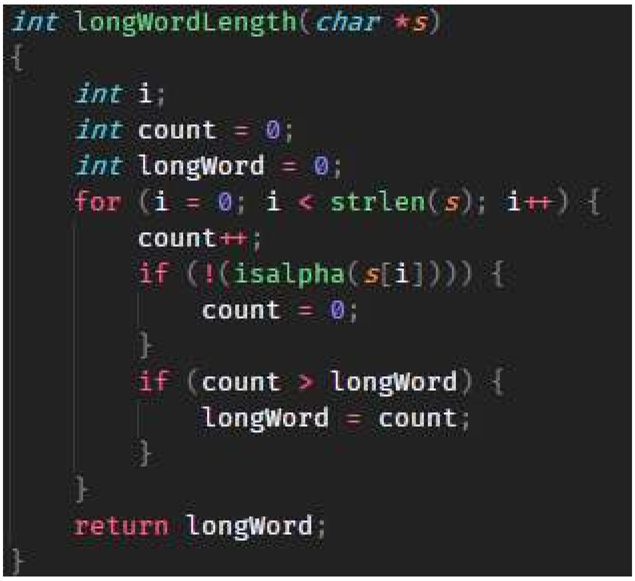
```
void insertChar(char *str1, char *str2, char ch)
{
    int i=0,j=0;
    while(1)
    {
      if((j+1)%4 == 0 && j != 0 && j>2)
      {
          str2[j] = ch;
          j++;
      }
      else
      {
           str2[j] = str1[i];
           i++;
           j++;
      }

      if(str1[i-1] == '\0')
      {
          break;
      }
    }
    str2[j]='\0';
}
```

2.locateFirstChar



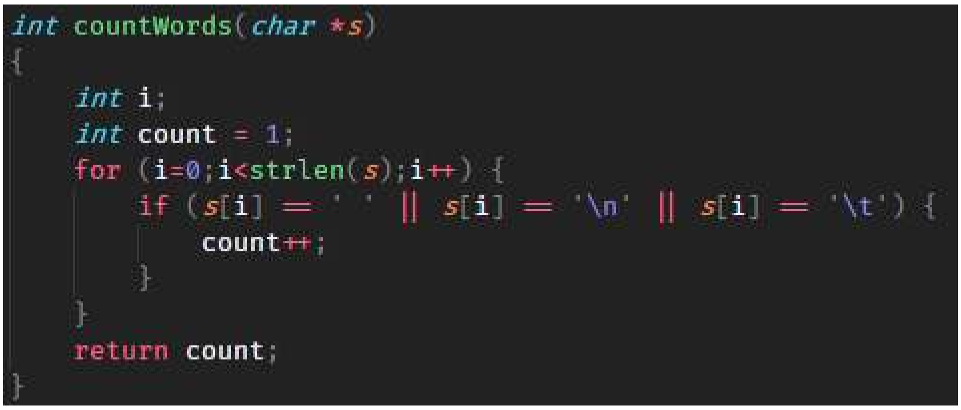
```
int locateFirstChar(char *str, char ch)
{
  int i;
  for(i=0;i<strlen(str);i++){
    if(str[i]==ch){
      return i;
    }
  }
  return -1;
}
```

3.longWordLength



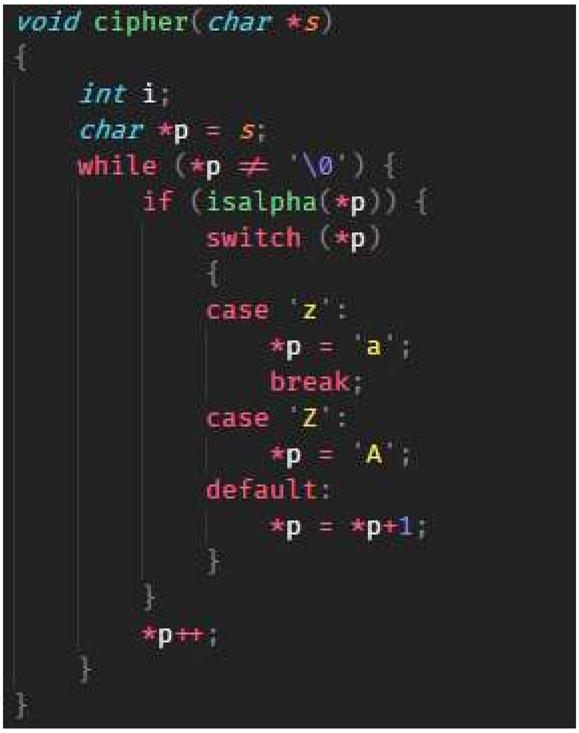
```c
int longWordLength(char *s)
{
    int i;
    int count = 0;
    int longWord = 0;
    for (i = 0; i < strlen(s); i++) {
        count++;
        if (!(isalpha(s[i]))) {
            count = 0;
        }
        if (count > longWord) {
            longWord = count;
        }
    }
    return longWord;
}
```

4.countWords



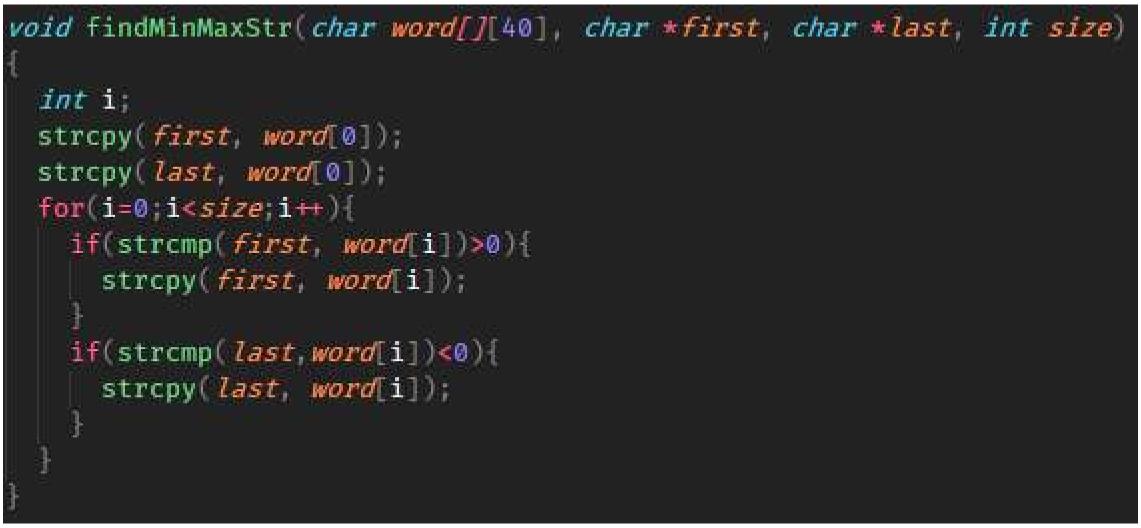
```c
int countWords(char *s)
{
    int i;
    int count = 1;
    for (i=0;i<strlen(s);i++) {
        if (s[i] == ' ' || s[i] == '\n' || s[i] == '\t') {
            count++;
        }
    }
    return count;
}
```

5.cipherText



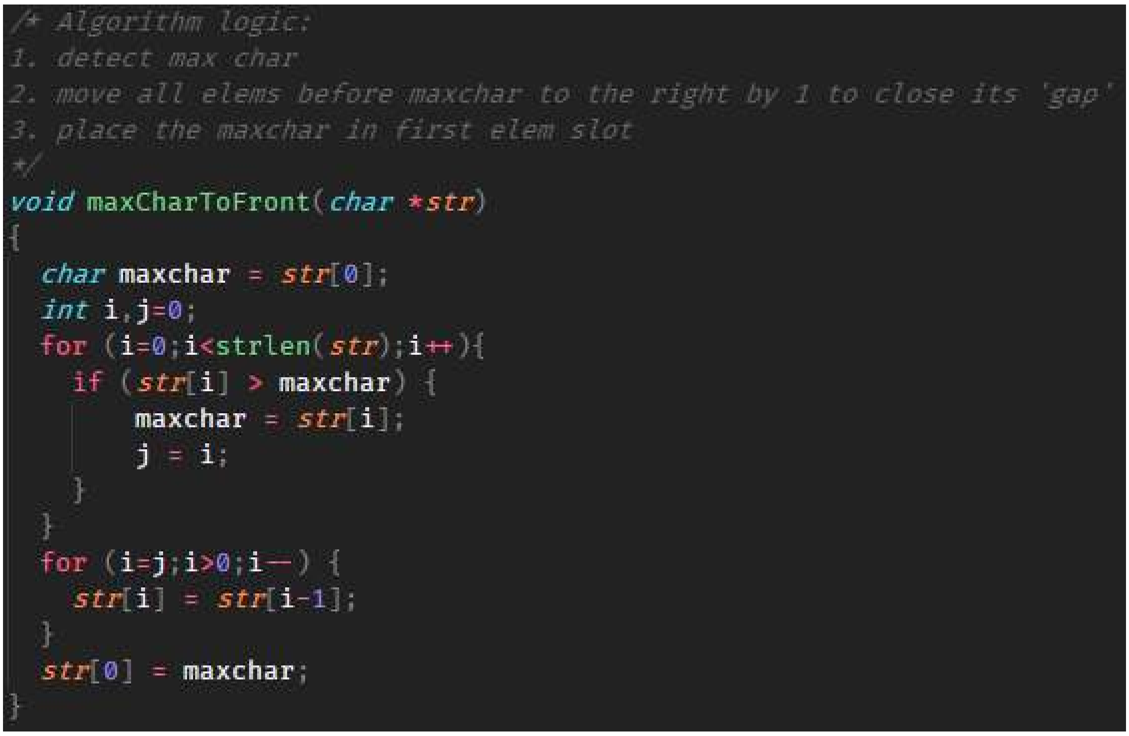
```c
void cipher(char *s)
{
    int i;
    char *p = s;
    while (*p ≠ '\0') {
        if (isalpha(*p)) {
            switch (*p)
            {
            case 'z':
                *p = 'a';
                break;
            case 'Z':
                *p = 'A';
            default:
                *p = *p+1;
            }
        }
        *p++;
    }
}
```

6.findMinMaxStr



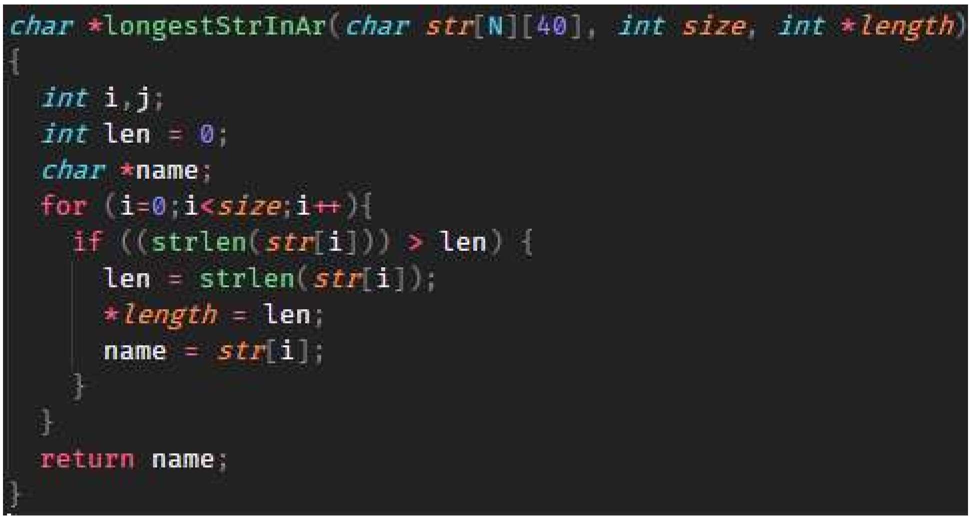
```c
void findMinMaxStr(char word[][40], char *first, char *last, int size)
{
  int i;
  strcpy(first, word[0]);
  strcpy(last, word[0]);
  for(i=0;i<size;i++){
    if(strcmp(first, word[i])>0){
      strcpy(first, word[i]);
    }
    if(strcmp(last,word[i])<0){
      strcpy(last, word[i]);
    }
  }
}
```

7.maxCharToFront



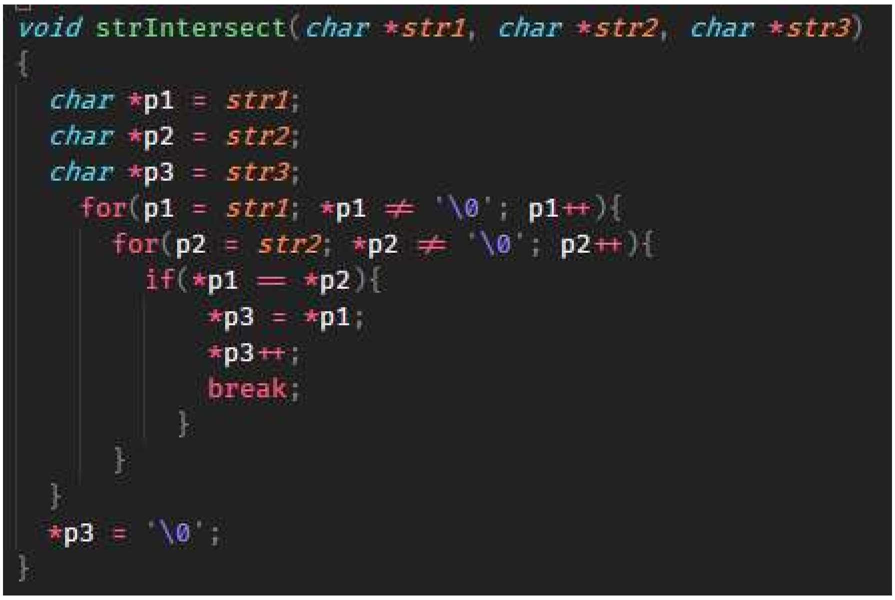
```
/* Algorithm logic:
1. detect max char
2. move all elems before maxchar to the right by 1 to close its 'gap'
3. place the maxchar in first elem slot
*/
void maxCharToFront(char *str)
{
  char maxchar = str[0];
  int i,j=0;
  for (i=0;i<strlen(str);i++){
    if (str[i] > maxchar) {
        maxchar = str[i];
        j = i;
    }
  }
  for (i=j;i>0;i--) {
    str[i] = str[i-1];
  }
  str[0] = maxchar;
}
```

8.longestStrInAr



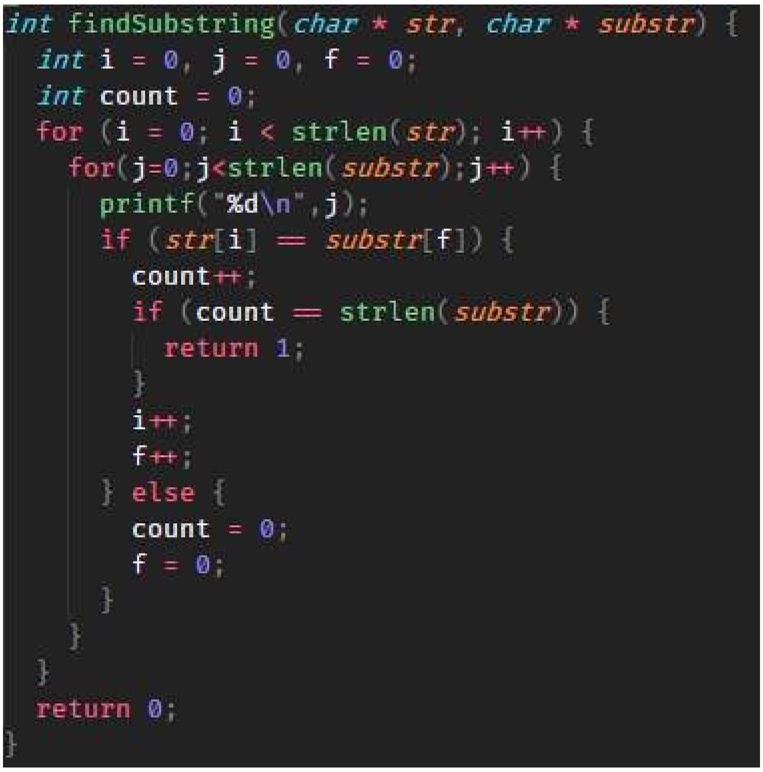
```
char *longestStrInAr(char str[N][40], int size, int *length)
{
  int i,j;
  int len = 0;
  char *name;
  for (i=0;i<size;i++){
    if ((strlen(str[i])) > len) {
      len = strlen(str[i]);
      *length = len;
      name = str[i];
    }
  }
  return name;
}
```

9.strIntersect



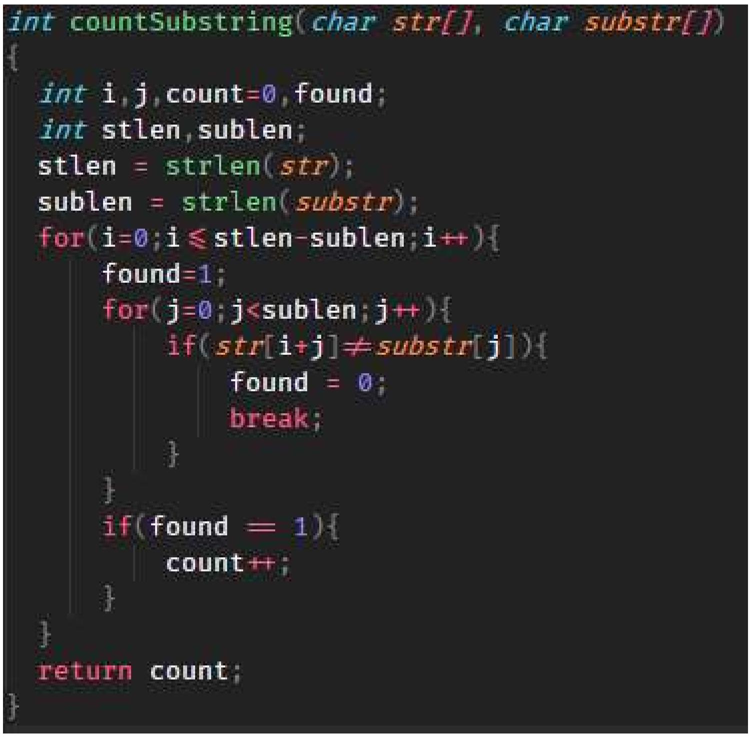
```
void strIntersect(char *str1, char *str2, char *str3)
{
  char *p1 = str1;
  char *p2 = str2;
  char *p3 = str3;
    for(p1 = str1; *p1 != '\0'; p1++){
      for(p2 = str2; *p2 != '\0'; p2++){
        if(*p1 == *p2){
            *p3 = *p1;
            *p3++;
            break;
          }
      }
  }
  *p3 = '\0';
}
```

10.findSubstring



```
int findSubstring(char * str, char * substr) {
  int i = 0, j = 0, f = 0;
  int count = 0;
  for (i = 0; i < strlen(str); i++) {
    for(j=0;j<strlen(substr);j++) {
      printf("%d\n",j);
      if (str[i] == substr[f]) {
        count++;
        if (count == strlen(substr)) {
          return 1;
        }
        i++;
        f++;
      } else {
        count = 0;
        f = 0;
      }
    }
  }
  return 0;
}
```

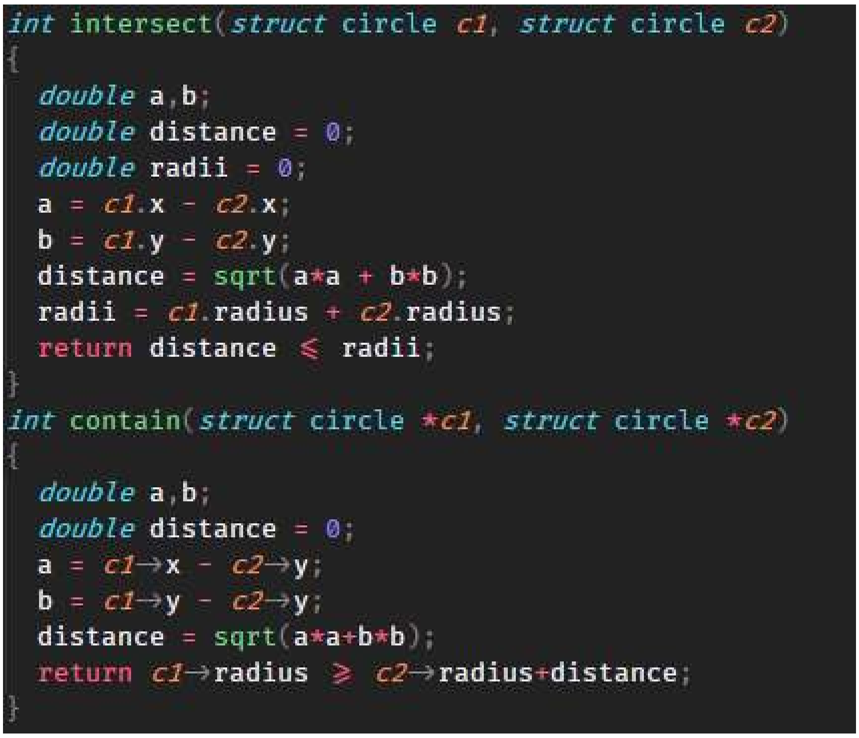
11.countSubstring



```c
int countSubstring(char str[], char substr[])
{
  int i,j,count=0,found;
  int stlen,sublen;
  stlen = strlen(str);
  sublen = strlen(substr);
  for(i=0;i<=stlen-sublen;i++){
      found=1;
      for(j=0;j<sublen;j++){
          if(str[i+j]!=substr[j]){
              found = 0;
              break;
          }
      }
      if(found == 1){
          count++;
      }
  }
  return count;
}
```

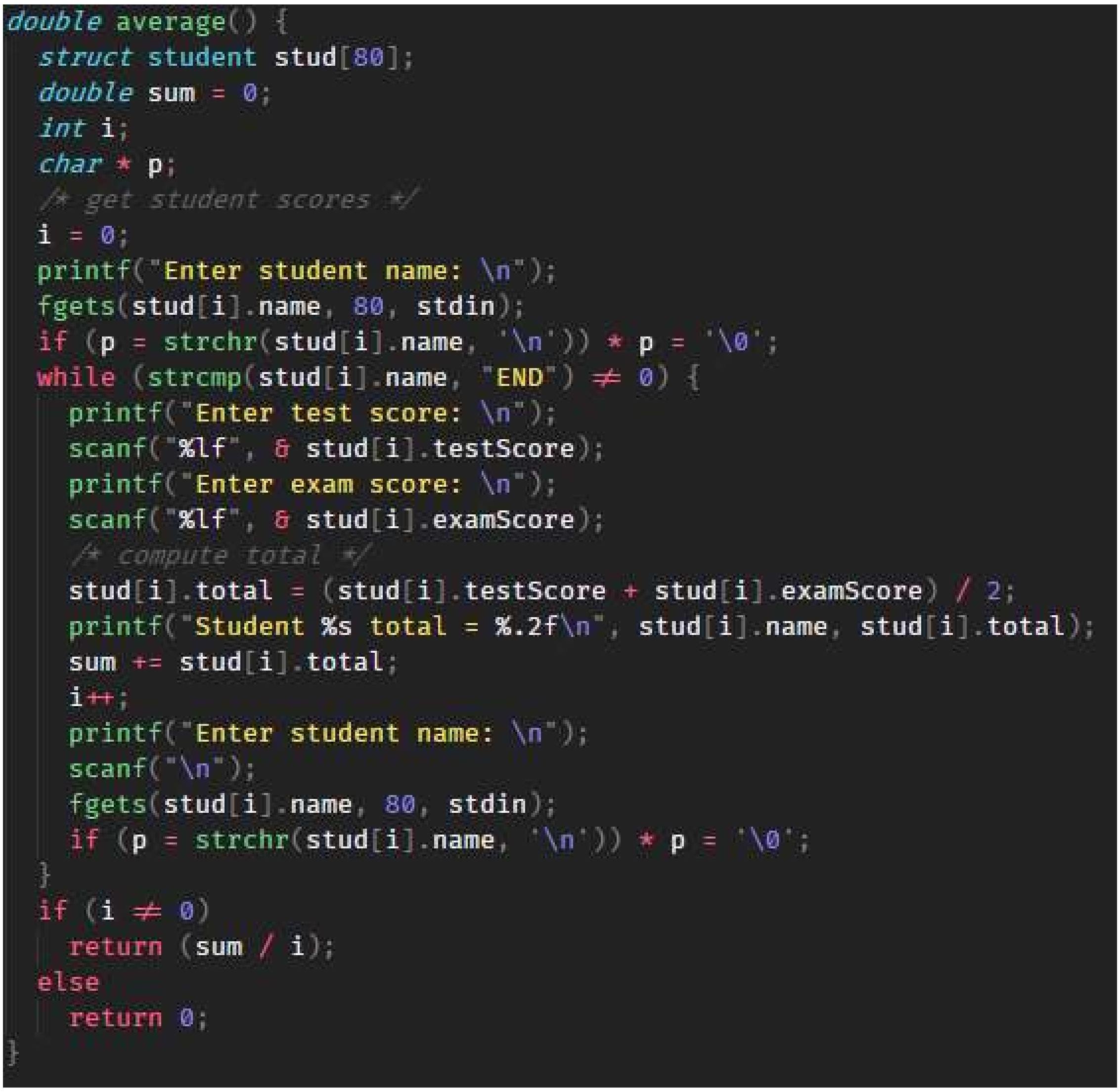
********6********

T1 computeCircle



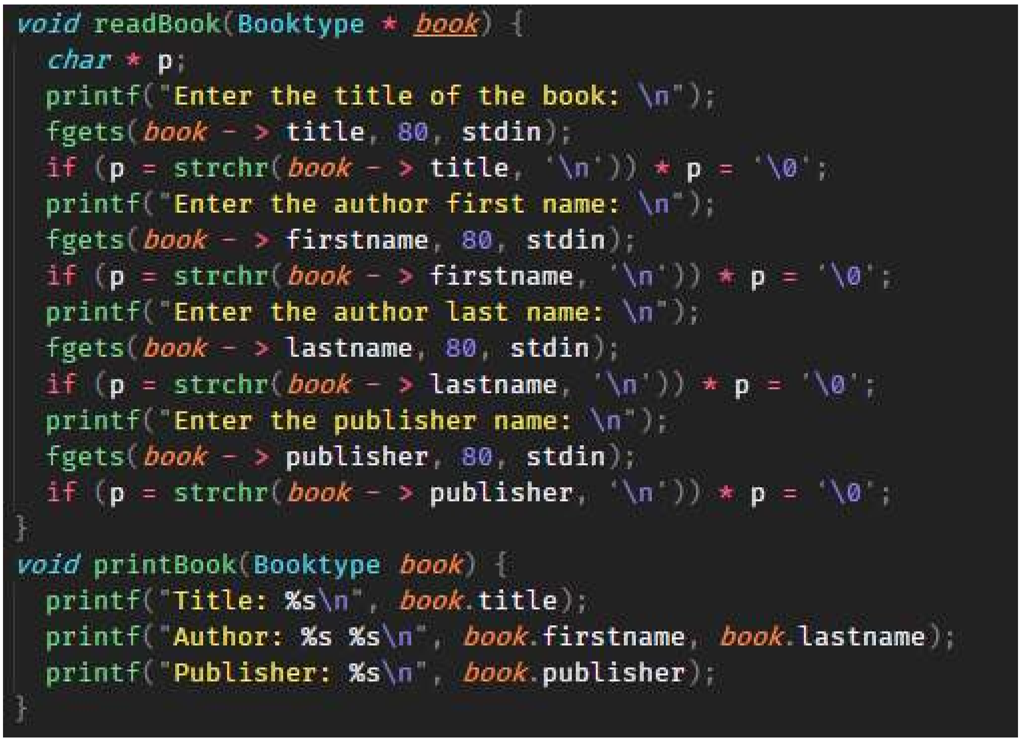
```c
int intersect(struct circle c1, struct circle c2)
{
  double a,b;
  double distance = 0;
  double radii = 0;
  a = c1.x - c2.x;
  b = c1.y - c2.y;
  distance = sqrt(a*a + b*b);
  radii = c1.radius + c2.radius;
  return distance <= radii;
}
int contain(struct circle *c1, struct circle *c2)
{
  double a,b;
  double distance = 0;
  a = c1->x - c2->y;
  b = c1->y - c2->y;
  distance = sqrt(a*a+b*b);
  return c1->radius >= c2->radius+distance;
}
```

T2 computeAverage



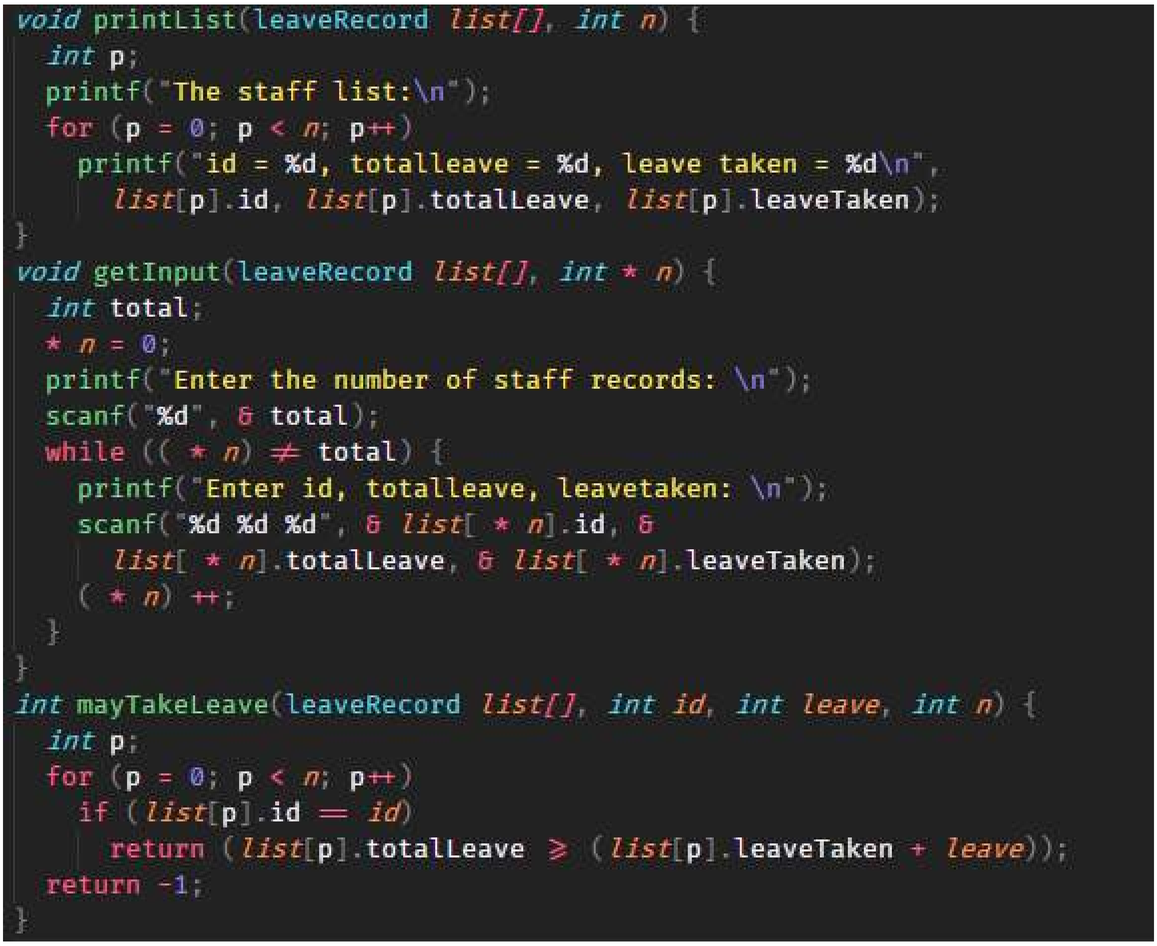
```
double average() {
  struct student stud[80];
  double sum = 0;
  int i;
  char * p;
  /* get student scores */
  i = 0;
  printf("Enter student name: \n");
  fgets(stud[i].name, 80, stdin);
  if (p = strchr(stud[i].name, '\n')) * p = '\0';
  while (strcmp(stud[i].name, "END") ≠ 0) {
    printf("Enter test score: \n");
    scanf("%lf", & stud[i].testScore);
    printf("Enter exam score: \n");
    scanf("%lf", & stud[i].examScore);
    /* compute total */
    stud[i].total = (stud[i].testScore + stud[i].examScore) / 2;
    printf("Student %s total = %.2f\n", stud[i].name, stud[i].total);
    sum += stud[i].total;
    i++;
    printf("Enter student name: \n");
    scanf("\n");
    fgets(stud[i].name, 80, stdin);
    if (p = strchr(stud[i].name, '\n')) * p = '\0';
  }
  if (i ≠ 0)
    return (sum / i);
  else
    return 0;
}
```

T3 book



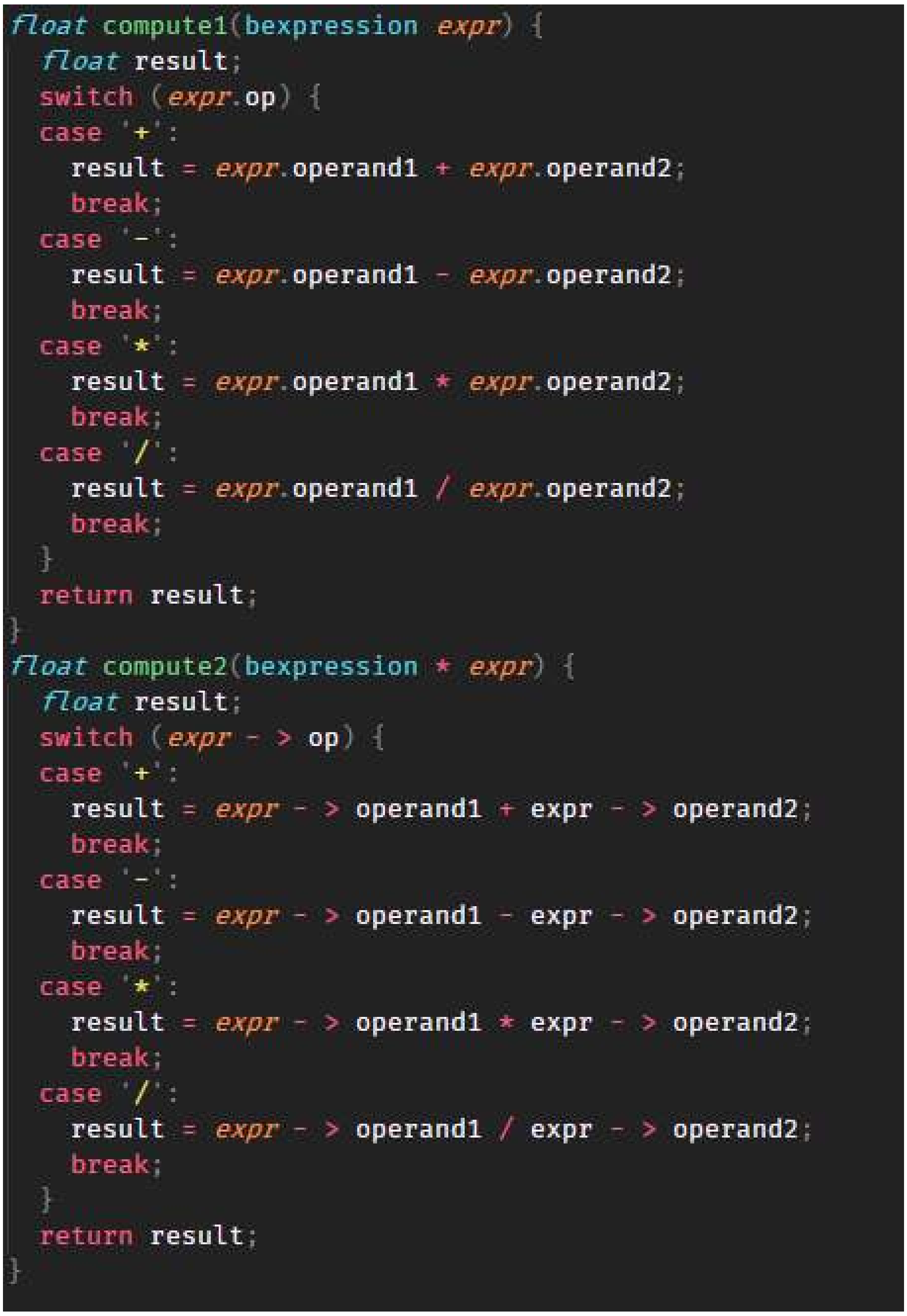
```c
void readBook(Booktype * book) {
  char * p;
  printf("Enter the title of the book: \n");
  fgets(book - > title, 80, stdin);
  if (p = strchr(book - > title, '\n')) * p = '\0';
  printf("Enter the author first name: \n");
  fgets(book - > firstname, 80, stdin);
  if (p = strchr(book - > firstname, '\n')) * p = '\0';
  printf("Enter the author last name: \n");
  fgets(book - > lastname, 80, stdin);
  if (p = strchr(book - > lastname, '\n')) * p = '\0';
  printf("Enter the publisher name: \n");
  fgets(book - > publisher, 80, stdin);
  if (p = strchr(book - > publisher, '\n')) * p = '\0';
}
void printBook(Booktype book) {
  printf("Title: %s\n", book.title);
  printf("Author: %s %s\n", book.firstname, book.lastname);
  printf("Publisher: %s\n", book.publisher);
}
```

T4 mayTakeLeave



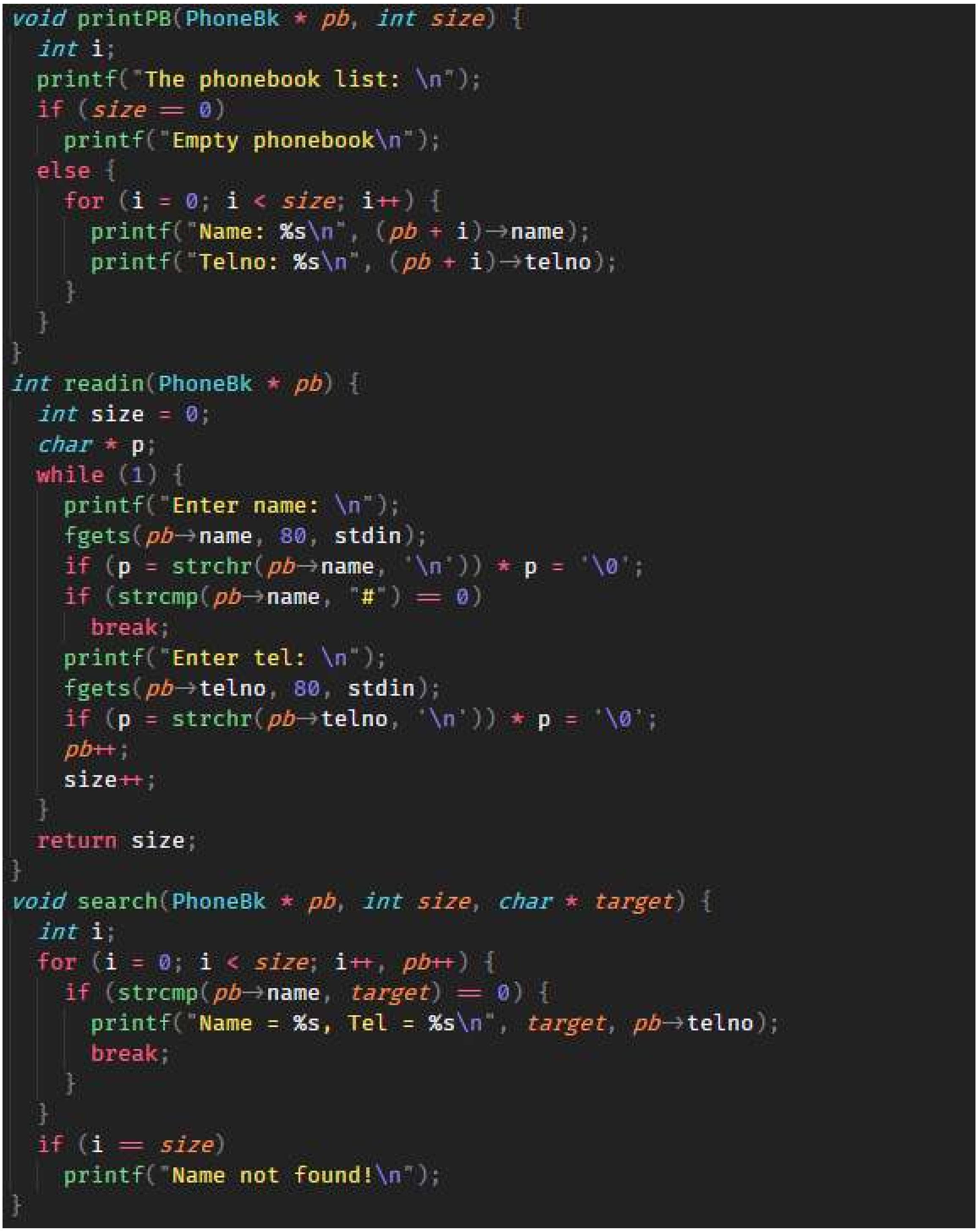
```c
void printList(leaveRecord list[], int n) {
  int p;
  printf("The staff list:\n");
  for (p = 0; p < n; p++)
    printf("id = %d, totalleave = %d, leave taken = %d\n",
      list[p].id, list[p].totalLeave, list[p].leaveTaken);
}
void getInput(leaveRecord list[], int * n) {
  int total;
  * n = 0;
  printf("Enter the number of staff records: \n");
  scanf("%d", & total);
  while (( * n) != total) {
    printf("Enter id, totalleave, leavetaken: \n");
    scanf("%d %d %d", & list[ * n].id, &
      list[ * n].totalLeave, & list[ * n].leaveTaken);
    ( * n) ++;
  }
}
int mayTakeLeave(leaveRecord list[], int id, int leave, int n) {
  int p;
  for (p = 0; p < n; p++)
    if (list[p].id == id)
      return (list[p].totalLeave >= (list[p].leaveTaken + leave));
  return -1;
}
```

L1 computeExp



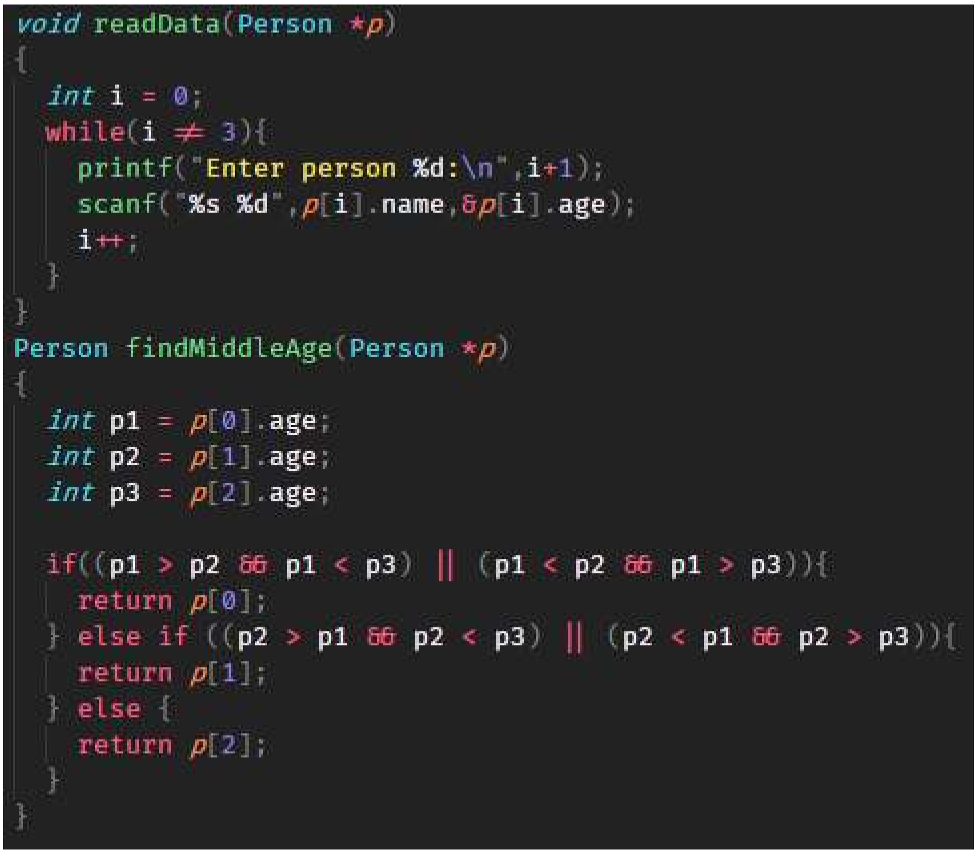
```
float compute1(bexpression expr) {
  float result;
  switch (expr.op) {
  case '+':
    result = expr.operand1 + expr.operand2;
    break;
  case '-':
    result = expr.operand1 - expr.operand2;
    break;
  case '*':
    result = expr.operand1 * expr.operand2;
    break;
  case '/':
    result = expr.operand1 / expr.operand2;
    break;
  }
  return result;
}
float compute2(bexpression * expr) {
  float result;
  switch (expr - > op) {
  case '+':
    result = expr - > operand1 + expr - > operand2;
    break;
  case '-':
    result = expr - > operand1 - expr - > operand2;
    break;
  case '*':
    result = expr - > operand1 * expr - > operand2;
    break;
  case '/':
    result = expr - > operand1 / expr - > operand2;
    break;
  }
  return result;
}
```

L2 phoneBook



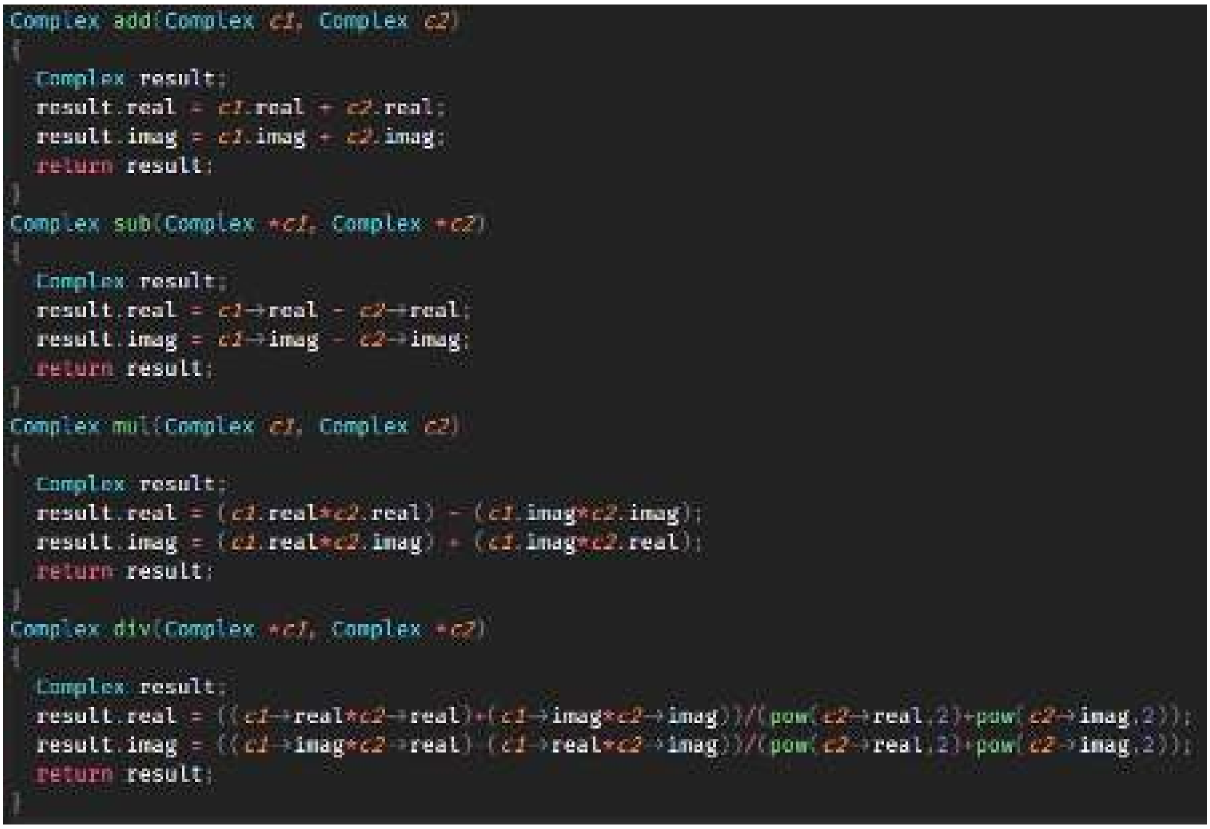
```c
void printPB(PhoneBk * pb, int size) {
  int i;
  printf("The phonebook list: \n");
  if (size == 0)
    printf("Empty phonebook\n");
  else {
    for (i = 0; i < size; i++) {
      printf("Name: %s\n", (pb + i)->name);
      printf("Telno: %s\n", (pb + i)->telno);
    }
  }
}
int readin(PhoneBk * pb) {
  int size = 0;
  char * p;
  while (1) {
    printf("Enter name: \n");
    fgets(pb->name, 80, stdin);
    if (p = strchr(pb->name, '\n')) * p = '\0';
    if (strcmp(pb->name, "#") == 0)
      break;
    printf("Enter tel: \n");
    fgets(pb->telno, 80, stdin);
    if (p = strchr(pb->telno, '\n')) * p = '\0';
    pb++;
    size++;
  }
  return size;
}
void search(PhoneBk * pb, int size, char * target) {
  int i;
  for (i = 0; i < size; i++, pb++) {
    if (strcmp(pb->name, target) == 0) {
      printf("Name = %s, Tel = %s\n", target, pb->telno);
      break;
    }
  }
  if (i == size)
    printf("Name not found!\n");
}
```

1.findMiddleAge



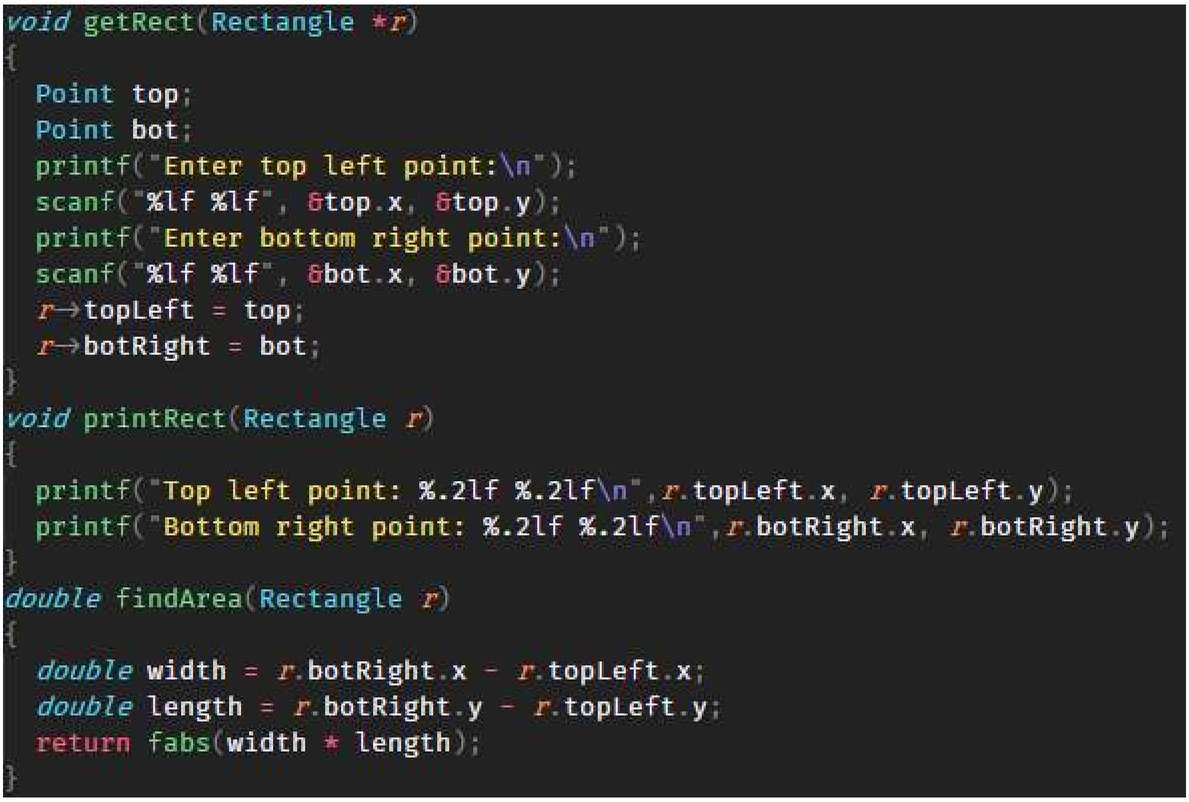
```
void readData(Person *p)
{
  int i = 0;
  while(i ≠ 3){
    printf("Enter person %d:\n",i+1);
    scanf("%s %d",p[i].name,&p[i].age);
    i++;
  }
}
Person findMiddleAge(Person *p)
{
  int p1 = p[0].age;
  int p2 = p[1].age;
  int p3 = p[2].age;

  if((p1 > p2 && p1 < p3) || (p1 < p2 && p1 > p3)){
    return p[0];
  } else if ((p2 > p1 && p2 < p3) || (p2 < p1 && p2 > p3)){
    return p[1];
  } else {
    return p[2];
  }
}
```

2.complexNumber



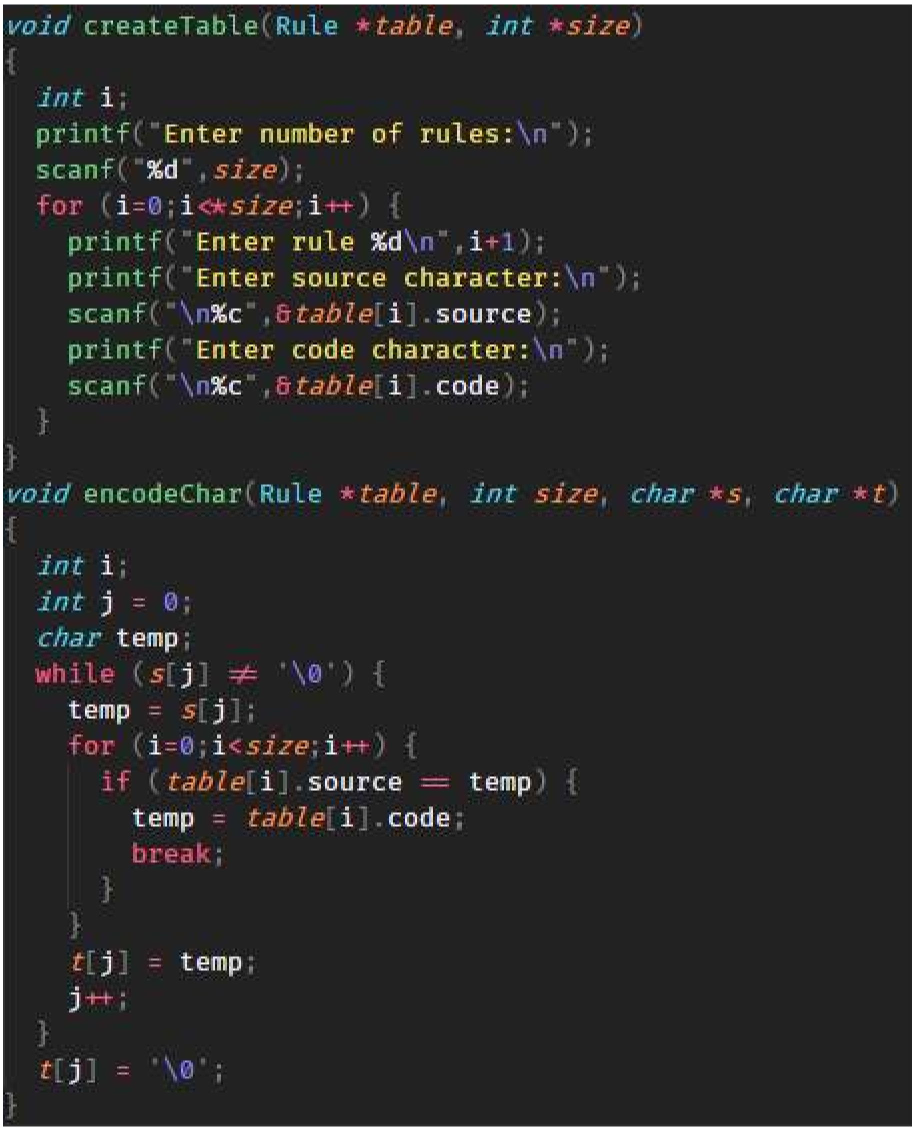
```
Complex add(Complex c1, Complex c2)
{
  Complex result;
  result.real = c1.real + c2.real;
  result.imag = c1.imag + c2.imag;
  return result;
}
Complex sub(Complex *c1, Complex *c2)
{
  Complex result;
  result.real = c1→real - c2→real;
  result.imag = c1→imag - c2→imag;
  return result;
}
Complex mul(Complex c1, Complex c2)
{
  Complex result;
  result.real = (c1.real*c2.real) - (c1.imag*c2.imag);
  result.imag = (c1.real*c2.imag) + (c1.imag*c2.real);
  return result;
}
Complex div(Complex *c1, Complex *c2)
{
  Complex result;
  result.real = ((c1→real*c2→real)+(c1→imag*c2→imag))/(pow(c2→real,2)+pow(c2→imag,2));
  result.imag = ((c1→imag*c2→real)-(c1→real*c2→imag))/(pow(c2→real,2)+pow(c2→imag,2));
  return result;
}
```

3.rectangle



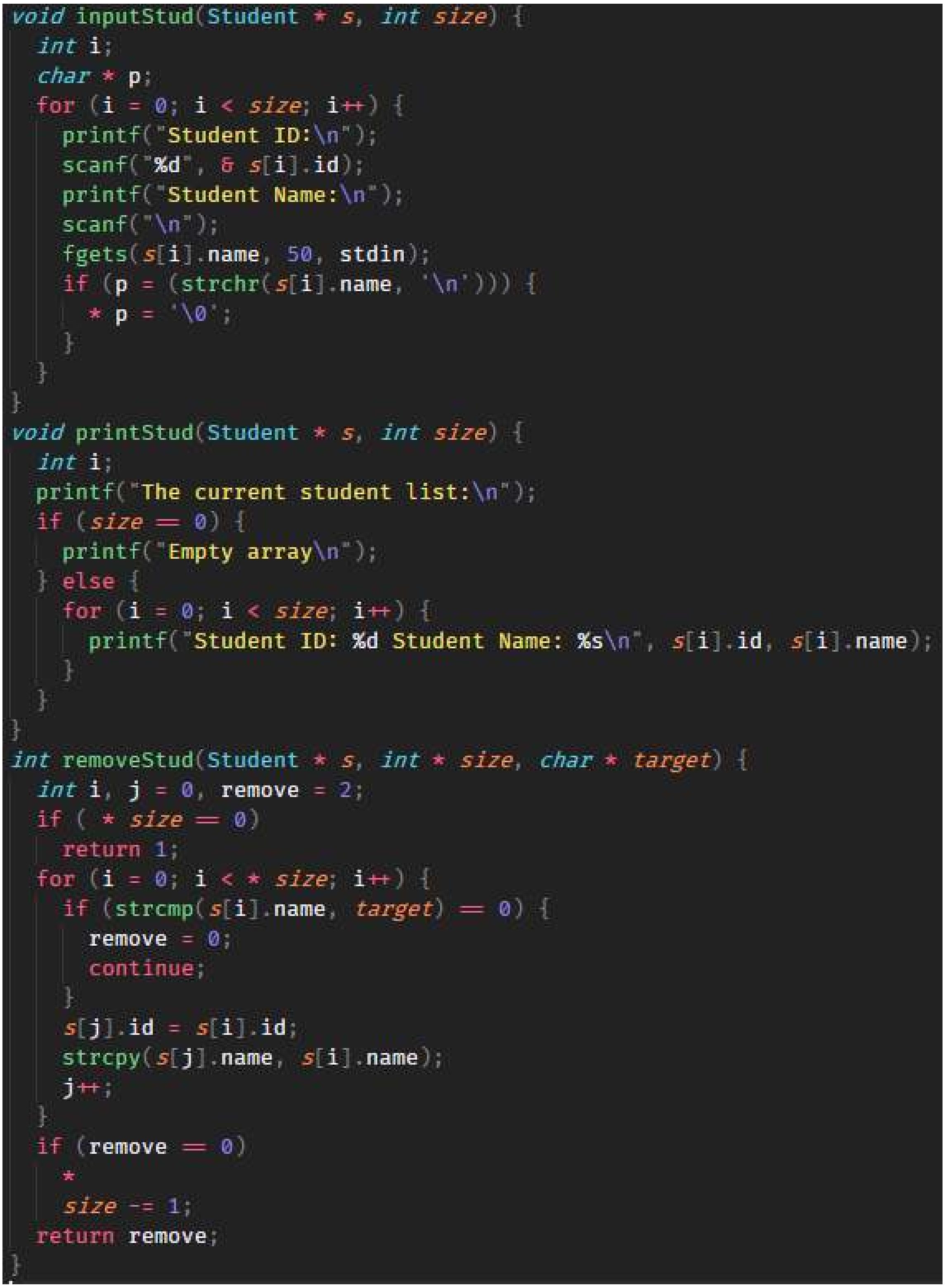
```
void getRect(Rectangle *r)
{
  Point top;
  Point bot;
  printf("Enter top left point:\n");
  scanf("%lf %lf", &top.x, &top.y);
  printf("Enter bottom right point:\n");
  scanf("%lf %lf", &bot.x, &bot.y);
  r->topLeft = top;
  r->botRight = bot;
}
void printRect(Rectangle r)
{
  printf("Top left point: %.2lf %.2lf\n",r.topLeft.x, r.topLeft.y);
  printf("Bottom right point: %.2lf %.2lf\n",r.botRight.x, r.botRight.y);
}
double findArea(Rectangle r)
{
  double width = r.botRight.x - r.topLeft.x;
  double length = r.botRight.y - r.topLeft.y;
  return fabs(width * length);
}
```

4.encodeChar



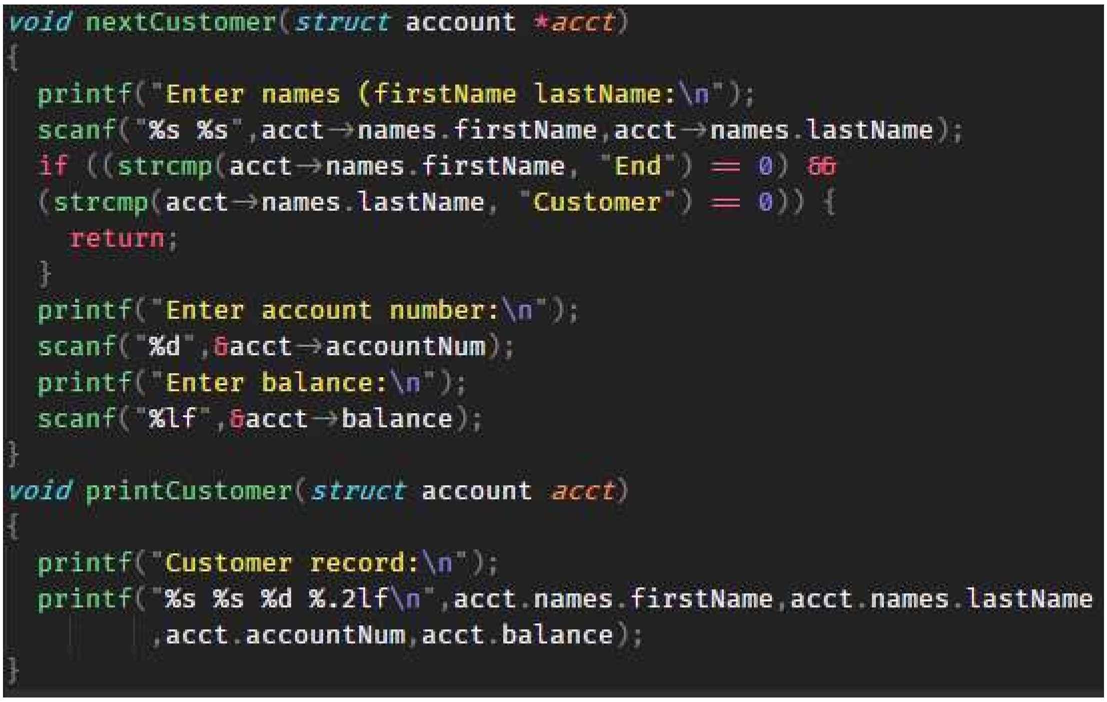
```
void createTable(Rule *table, int *size)
{
  int i;
  printf("Enter number of rules:\n");
  scanf("%d",size);
  for (i=0;i<*size;i++) {
    printf("Enter rule %d\n",i+1);
    printf("Enter source character:\n");
    scanf("\n%c",&table[i].source);
    printf("Enter code character:\n");
    scanf("\n%c",&table[i].code);
  }
}
void encodeChar(Rule *table, int size, char *s, char *t)
{
  int i;
  int j = 0;
  char temp;
  while (s[j] != '\0') {
    temp = s[j];
    for (i=0;i<size;i++) {
      if (table[i].source == temp) {
        temp = table[i].code;
        break;
      }
    }
    t[j] = temp;
    j++;
  }
  t[j] = '\0';
}
```

5.student



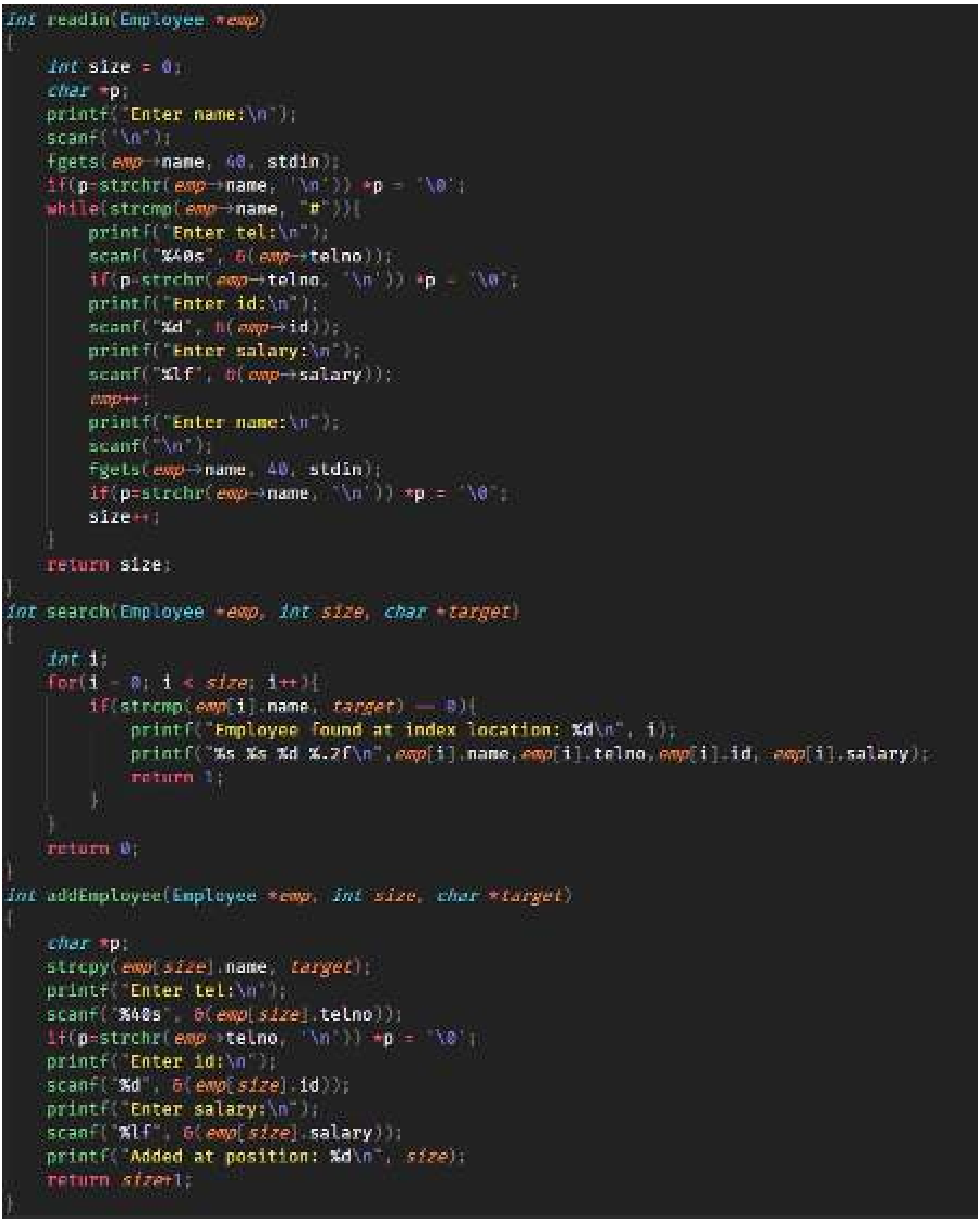
```
void inputStud(Student * s, int size) {
  int i;
  char * p;
  for (i = 0; i < size; i++) {
    printf("Student ID:\n");
    scanf("%d", & s[i].id);
    printf("Student Name:\n");
    scanf("\n");
    fgets(s[i].name, 50, stdin);
    if (p = (strchr(s[i].name, '\n'))) {
      * p = '\0';
    }
  }
}
void printStud(Student * s, int size) {
  int i;
  printf("The current student list:\n");
  if (size == 0) {
    printf("Empty array\n");
  } else {
    for (i = 0; i < size; i++) {
      printf("Student ID: %d Student Name: %s\n", s[i].id, s[i].name);
    }
  }
}
int removeStud(Student * s, int * size, char * target) {
  int i, j = 0, remove = 2;
  if ( * size == 0)
    return 1;
  for (i = 0; i < * size; i++) {
    if (strcmp(s[i].name, target) == 0) {
      remove = 0;
      continue;
    }
    s[j].id = s[i].id;
    strcpy(s[j].name, s[i].name);
    j++;
  }
  if (remove == 0)
    *
    size -= 1;
  return remove;
}
```

6.customer



```c
void nextCustomer(struct account *acct)
{
  printf("Enter names (firstName lastName:\n");
  scanf("%s %s",acct->names.firstName,acct->names.lastName);
  if ((strcmp(acct->names.firstName, "End") == 0) &&
  (strcmp(acct->names.lastName, "Customer") == 0)) {
    return;
  }
  printf("Enter account number:\n");
  scanf("%d",&acct->accountNum);
  printf("Enter balance:\n");
  scanf("%lf",&acct->balance);
}
void printCustomer(struct account acct)
{
  printf("Customer record:\n");
  printf("%s %s %d %.2lf\n",acct.names.firstName,acct.names.lastName
         ,acct.accountNum,acct.balance);
}
```

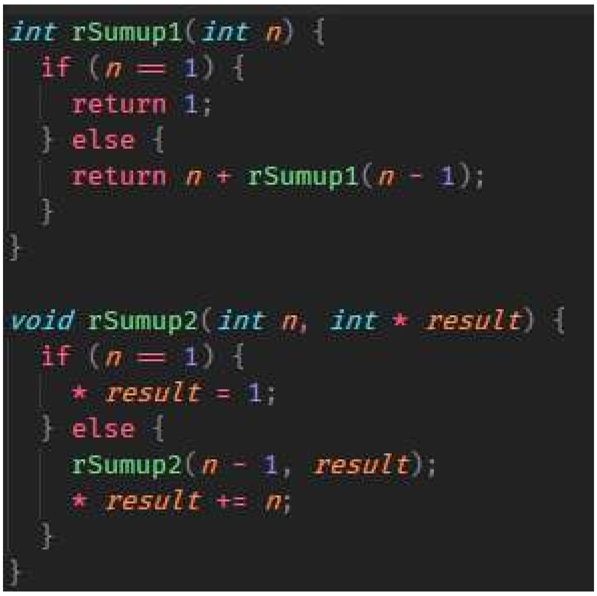
7.employee



```c
int readin(Employee *emp)
{
    int size = 0;
    char *p;
    printf("Enter name:\n");
    scanf("\n");
    fgets(emp->name, 40, stdin);
    if(p=strchr(emp->name, '\n')) *p = '\0';
    while(strcmp(emp->name, "#")){
        printf("Enter tel:\n");
        scanf("%40s", &(emp->telno));
        if(p=strchr(emp->telno, '\n')) *p = '\0';
        printf("Enter id:\n");
        scanf("%d", &(emp->id));
        printf("Enter salary:\n");
        scanf("%lf", &(emp->salary));
        emp++;
        printf("Enter name:\n");
        scanf("\n");
        fgets(emp->name, 40, stdin);
        if(p=strchr(emp->name, '\n')) *p = '\0';
        size++;
    }
    return size;
}
int search(Employee *emp, int size, char *target)
{
    int i;
    for(i = 0; i < size; i++){
        if(strcmp(emp[i].name, target) == 0){
            printf("Employee found at index location: %d\n", i);
            printf("%s %s %d %.2f\n",emp[i].name,emp[i].telno,emp[i].id, emp[i].salary);
            return 1;
        }
    }
    return 0;
}
int addEmployee(Employee *emp, int size, char *target)
{
    char *p;
    strcpy(emp[size].name, target);
    printf("Enter tel:\n");
    scanf("%40s", &(emp[size].telno));
    if(p=strchr(emp->telno, '\n')) *p = '\0';
    printf("Enter id:\n");
    scanf("%d", &(emp[size].id));
    printf("Enter salary:\n");
    scanf("%lf", &(emp[size].salary));
    printf("Added at position: %d\n", size);
    return size+1;
}
```

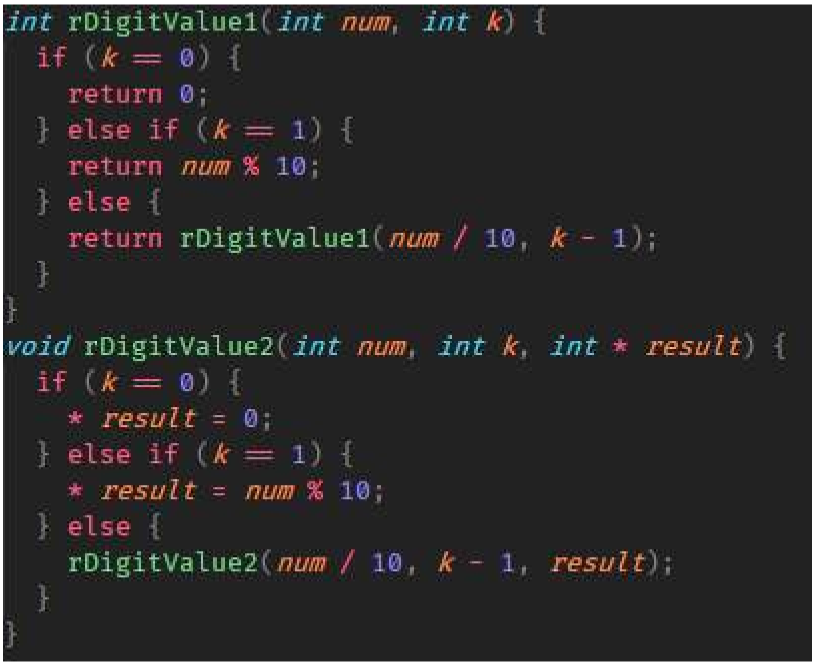
********7********

T1 rSumup



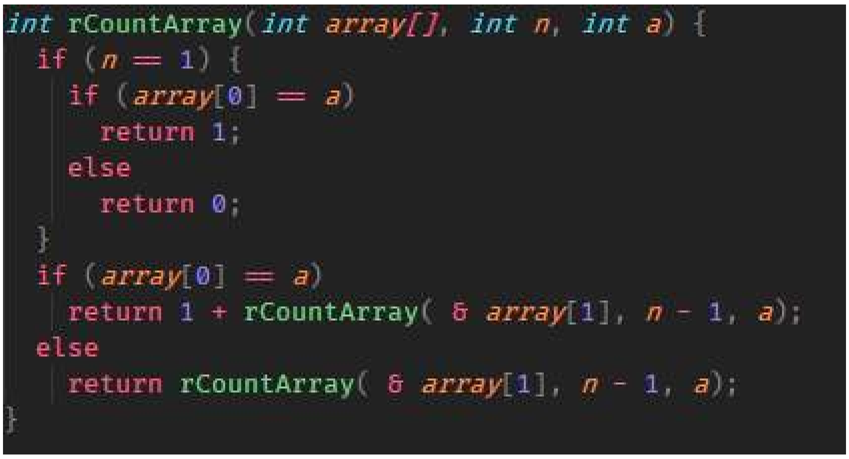
```
int rSumup1(int n) {
  if (n == 1) {
    return 1;
  } else {
    return n + rSumup1(n - 1);
  }
}

void rSumup2(int n, int * result) {
  if (n == 1) {
    * result = 1;
  } else {
    rSumup2(n - 1, result);
    * result += n;
  }
}
```

T2 rDigitValue



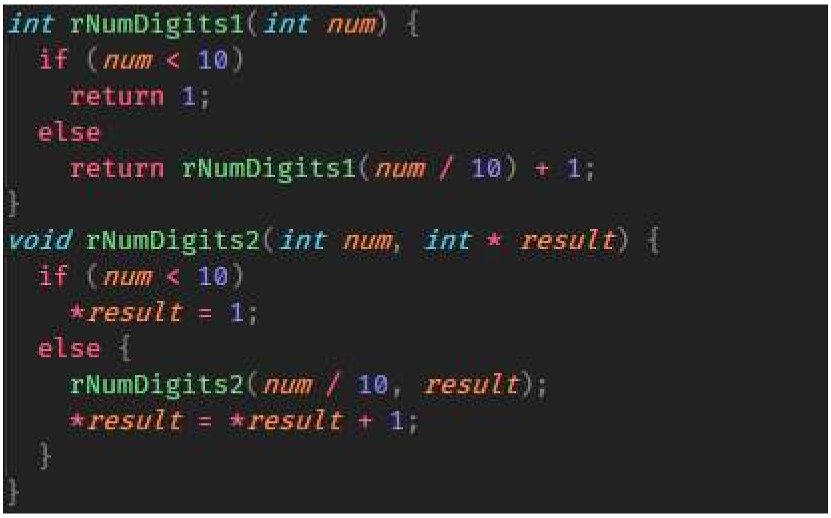
```
int rDigitValue1(int num, int k) {
  if (k == 0) {
    return 0;
  } else if (k == 1) {
    return num % 10;
  } else {
    return rDigitValue1(num / 10, k - 1);
  }
}
void rDigitValue2(int num, int k, int * result) {
  if (k == 0) {
    * result = 0;
  } else if (k == 1) {
    * result = num % 10;
  } else {
    rDigitValue2(num / 10, k - 1, result);
  }
}
```

T3 rCountArray



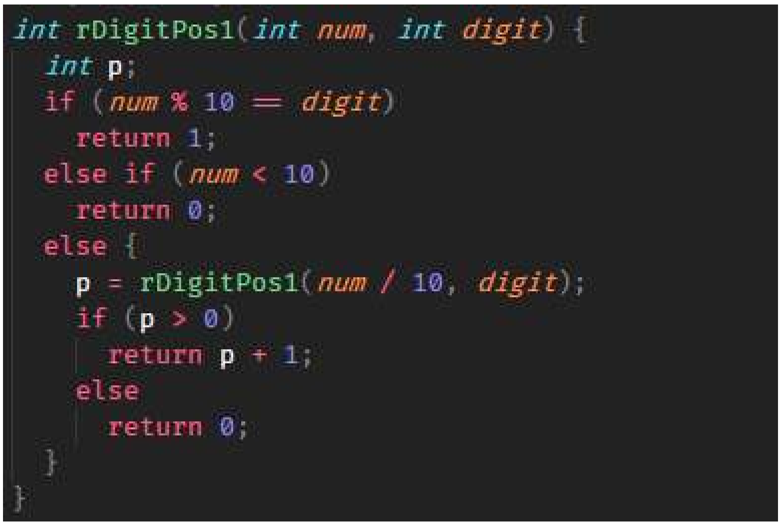
```
int rCountArray(int array[], int n, int a) {
  if (n == 1) {
    if (array[0] == a)
      return 1;
    else
      return 0;
  }
  if (array[0] == a)
    return 1 + rCountArray( & array[1], n - 1, a);
  else
    return rCountArray( & array[1], n - 1, a);
}
```

L1 rNumDigits



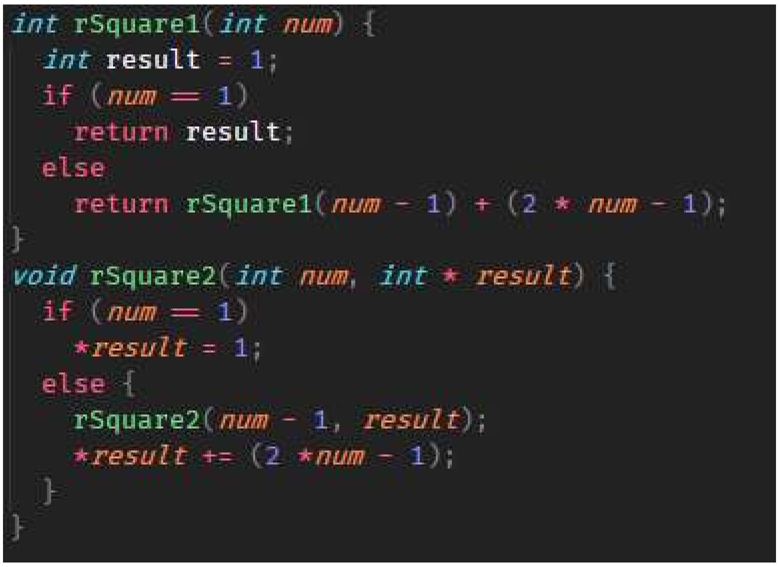
```
int rNumDigits1(int num) {
  if (num < 10)
    return 1;
  else
    return rNumDigits1(num / 10) + 1;
}
void rNumDigits2(int num, int * result) {
  if (num < 10)
    *result = 1;
  else {
    rNumDigits2(num / 10, result);
    *result = *result + 1;
  }
}
```

L2 rDigitPos



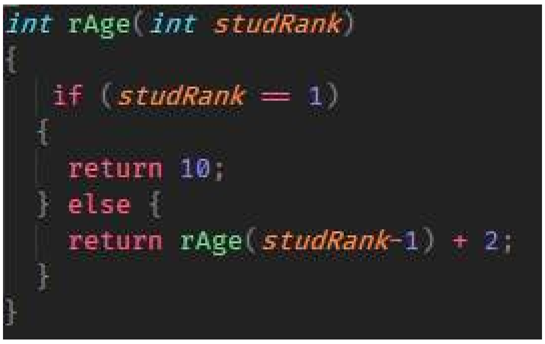
```
int rDigitPos1(int num, int digit) {
  int p;
  if (num % 10 == digit)
    return 1;
  else if (num < 10)
    return 0;
  else {
    p = rDigitPos1(num / 10, digit);
    if (p > 0)
      return p + 1;
    else
      return 0;
  }
}
```

L3 rSquare



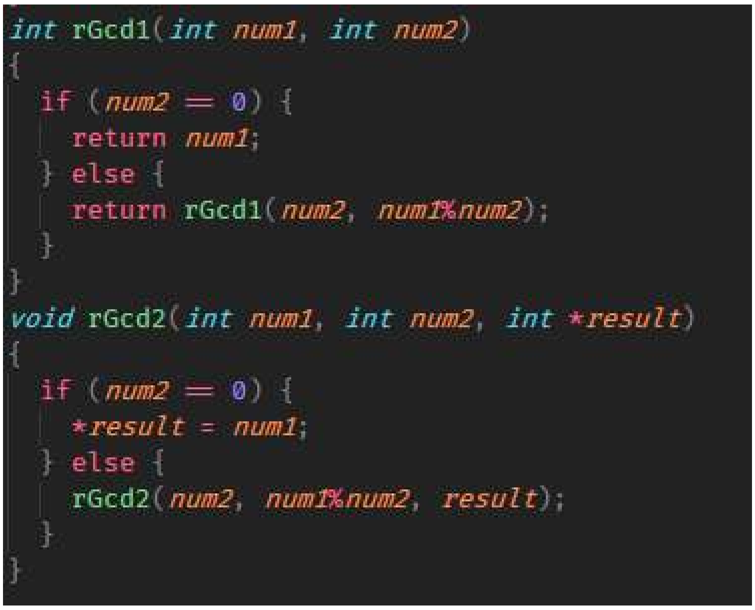
```c
int rSquare1(int num) {
  int result = 1;
  if (num == 1)
    return result;
  else
    return rSquare1(num - 1) + (2 * num - 1);
}
void rSquare2(int num, int * result) {
  if (num == 1)
    *result = 1;
  else {
    rSquare2(num - 1, result);
    *result += (2 *num - 1);
  }
}
```

1.rAge



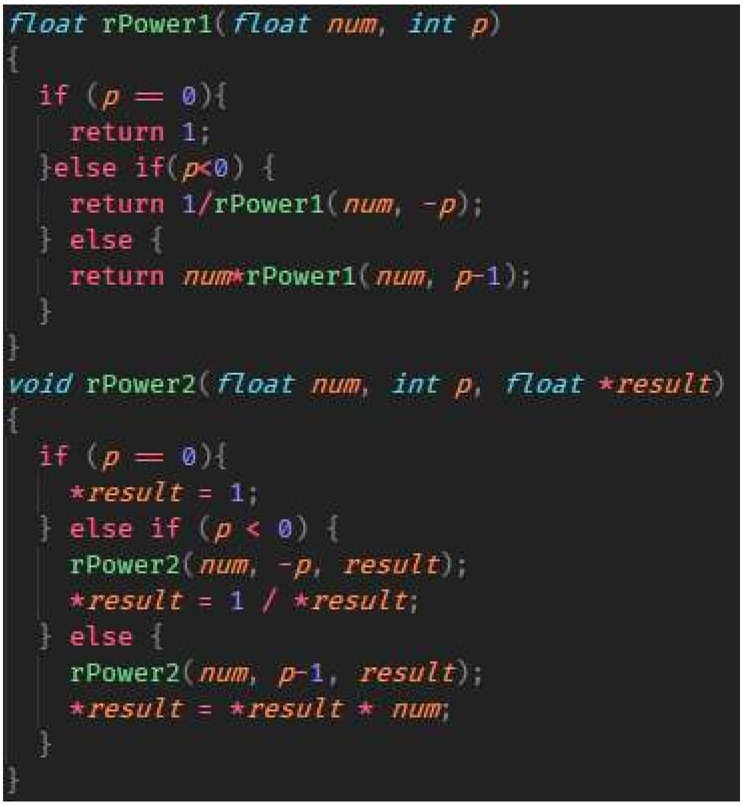
```c
int rAge(int studRank)
{
   if (studRank == 1)
  {
    return 10;
  } else {
    return rAge(studRank-1) + 2;
  }
}
```

2.rGcd



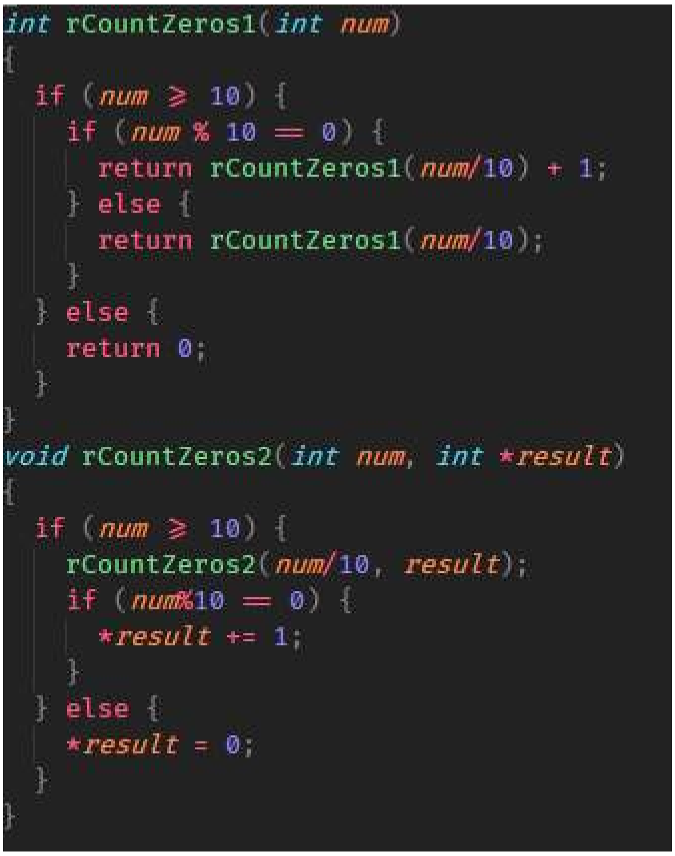
```c
int rGcd1(int num1, int num2)
{
  if (num2 == 0) {
    return num1;
  } else {
    return rGcd1(num2, num1%num2);
  }
}
void rGcd2(int num1, int num2, int *result)
{
  if (num2 == 0) {
    *result = num1;
  } else {
    rGcd2(num2, num1%num2, result);
  }
}
```

3.rPower



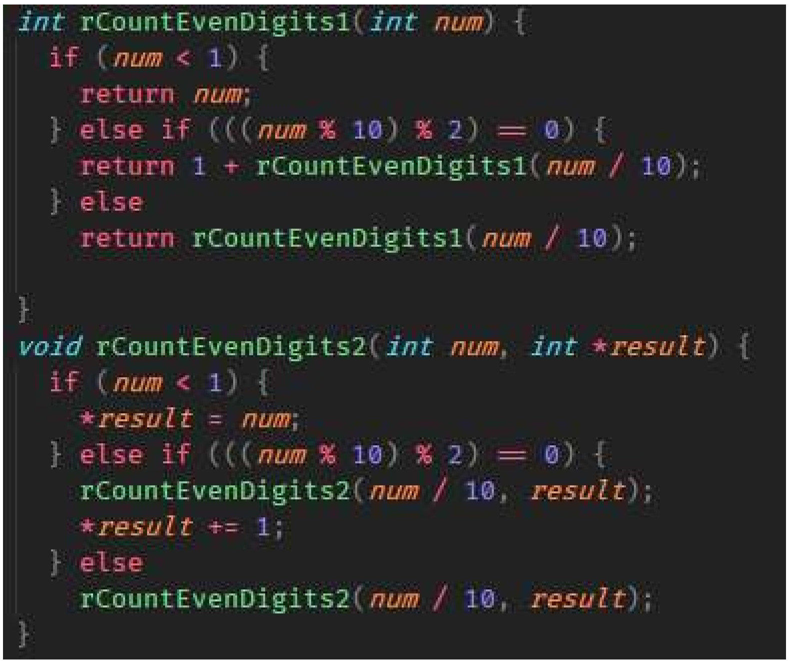
```
float rPower1(float num, int p)
{
  if (p == 0){
    return 1;
  }else if(p<0) {
    return 1/rPower1(num, -p);
  } else {
    return num*rPower1(num, p-1);
  }
}
void rPower2(float num, int p, float *result)
{
  if (p == 0){
    *result = 1;
  } else if (p < 0) {
    rPower2(num, -p, result);
    *result = 1 / *result;
  } else {
    rPower2(num, p-1, result);
    *result = *result * num;
  }
}
```

4.rCountZeros



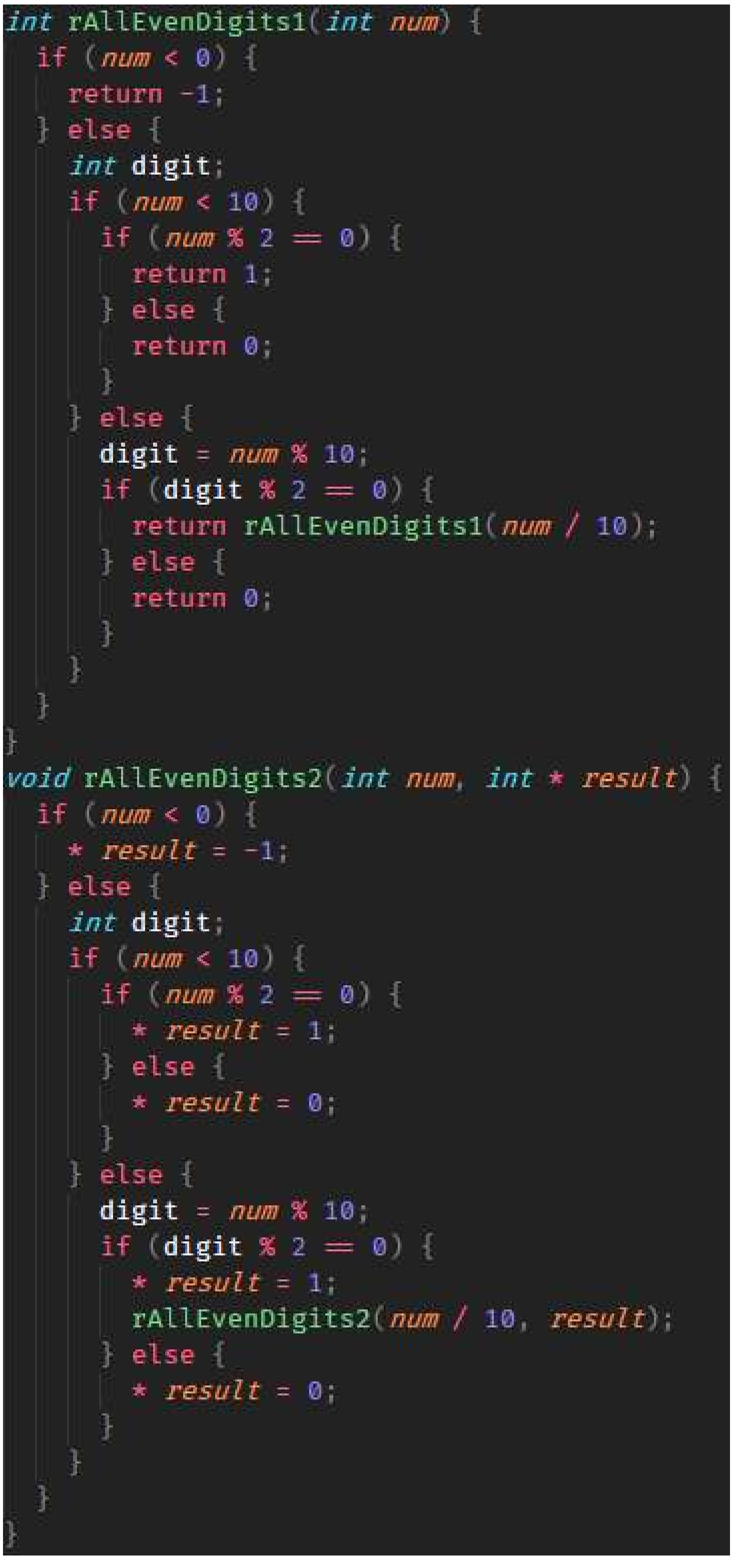
```
int rCountZeros1(int num)
{
  if (num >= 10) {
    if (num % 10 == 0) {
      return rCountZeros1(num/10) + 1;
    } else {
      return rCountZeros1(num/10);
    }
  } else {
    return 0;
  }
}
void rCountZeros2(int num, int *result)
{
  if (num >= 10) {
    rCountZeros2(num/10, result);
    if (num%10 == 0) {
      *result += 1;
    }
  } else {
    *result = 0;
  }
}
```

5.rCountEvenDigits



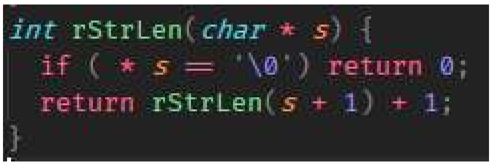
```
int rCountEvenDigits1(int num) {
  if (num < 1) {
    return num;
  } else if (((num % 10) % 2) == 0) {
    return 1 + rCountEvenDigits1(num / 10);
  } else
    return rCountEvenDigits1(num / 10);

}
void rCountEvenDigits2(int num, int *result) {
  if (num < 1) {
    *result = num;
  } else if (((num % 10) % 2) == 0) {
    rCountEvenDigits2(num / 10, result);
    *result += 1;
  } else
    rCountEvenDigits2(num / 10, result);
}
```

6.rAllEvenDigits



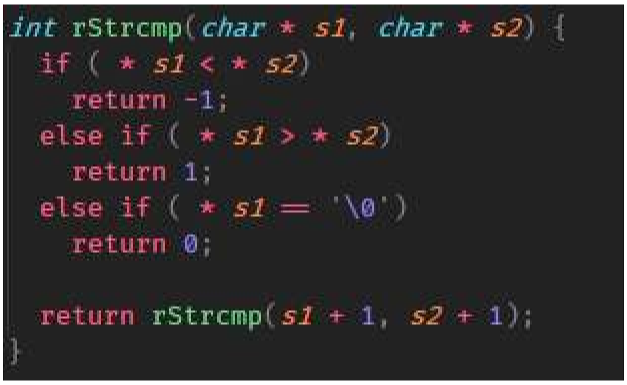
```
int rAllEvenDigits1(int num) {
  if (num < 0) {
    return -1;
  } else {
    int digit;
    if (num < 10) {
      if (num % 2 == 0) {
        return 1;
      } else {
        return 0;
      }
    } else {
      digit = num % 10;
      if (digit % 2 == 0) {
        return rAllEvenDigits1(num / 10);
      } else {
        return 0;
      }
    }
  }
}
void rAllEvenDigits2(int num, int * result) {
  if (num < 0) {
    * result = -1;
  } else {
    int digit;
    if (num < 10) {
      if (num % 2 == 0) {
        * result = 1;
      } else {
        * result = 0;
      }
    } else {
      digit = num % 10;
      if (digit % 2 == 0) {
        * result = 1;
        rAllEvenDigits2(num / 10, result);
      } else {
        * result = 0;
      }
    }
  }
}
```

7.rStrLen



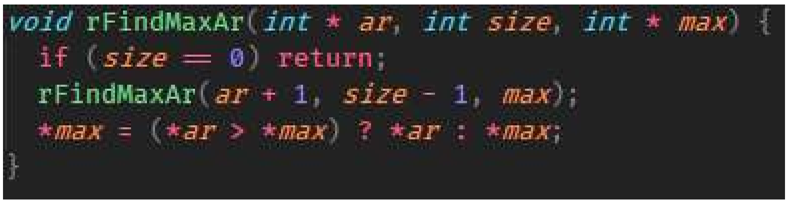
```
int rStrLen(char * s) {
  if ( * s == '\0') return 0;
  return rStrLen(s + 1) + 1;
}
```

8.rStrcmp



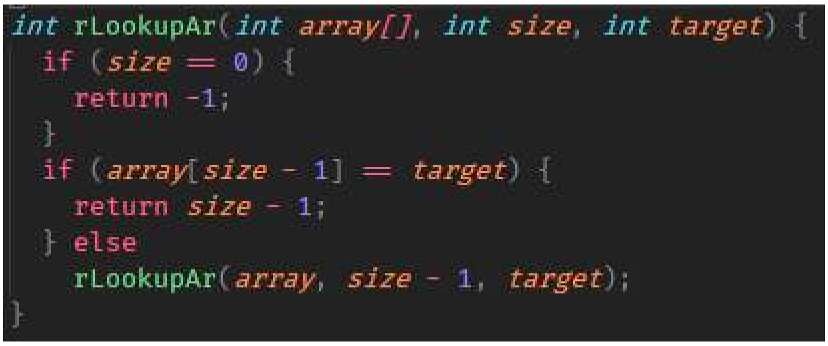
```
int rStrcmp(char * s1, char * s2) {
  if ( * s1 < * s2)
    return -1;
  else if ( * s1 > * s2)
    return 1;
  else if ( * s1 == '\0')
    return 0;

  return rStrcmp(s1 + 1, s2 + 1);
}
```

9.rFindMaxAr



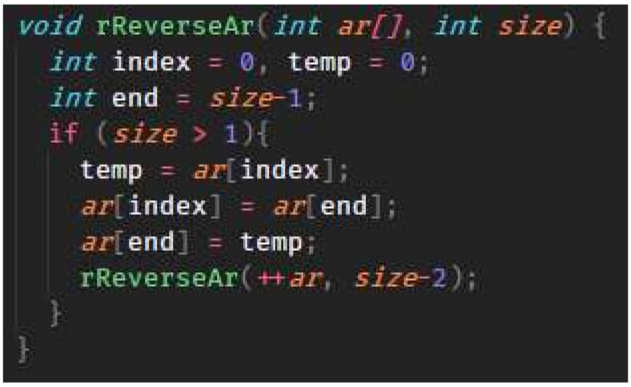
```
void rFindMaxAr(int * ar, int size, int * max) {
  if (size == 0) return;
  rFindMaxAr(ar + 1, size - 1, max);
  *max = (*ar > *max) ? *ar : *max;
}
```

10.rLookupAr



```
int rLookupAr(int array[], int size, int target) {
  if (size == 0) {
    return -1;
  }
  if (array[size - 1] == target) {
    return size - 1;
  } else
    rLookupAr(array, size - 1, target);
}
```

11.rReverseAr



```
void rReverseAr(int ar[], int size) {
  int index = 0, temp = 0;
  int end = size-1;
  if (size > 1){
    temp = ar[index];
    ar[index] = ar[end];
    ar[end] = temp;
    rReverseAr(++ar, size-2);
  }
}
```