MergeSort

```
#include<iostream>
using namespace std;

void merge(int A[],int low, int high){
 int i=low, j=(high+low)/2+1;      // indexy v poli A
 int k=0;                          // index v poli C
 int N=j, M=high+1;                // zarazky indexov i a j
 int* C=new int[high-low+1];

 while((i<N)&&(j<M)){
  while ((A[i]<=A[j])&&(i<N)){
      C[k++]=A[i++];
  }

  while (A[j]<=A[i]&&(j<M)){
      C[k++]=A[j++];
  }
 }

  for(;i<N;i++){  // ktore pole skoncilo testovat netreba, lebo cyklus prinajhorsom neprebehne
   C[k++]=A[i];
  }
  for(;j<M;j++){
   C[k++]=A[j];
  }

 for (int kk=low; kk<=high; kk++) A[kk]=C[kk-low];

 delete[] C;

}

void mergesort(int A[], int low, int high){
    if (low>=high) return;

    // rozdeluj
    int mid=(low+high)/2;

    //rekurzivne volanie na dva podpripady
    mergesort(A,low,mid);
    mergesort(A,mid+1,high);

    //panuj
    merge(A,low,high);
}

int main(void){
    int A[]={135,1,58,75,12,45,7,100,23,467};
    mergesort(A,0,9);
    for (int i=0; i<10; i++) cout<< A[i]<< " ";
}
```

QuickSort

```
int divide(int A[], int l, int r){
    int v=A[r]; // zvolim si pivot a odlozim si ho na kraj (alebo rovno zvolim krajny prvok)
    int i=l-1, j=r;
    int temp;
    while (i<j){
        while (A[++i]<v);
        while (A[--j]>v);
        if (i>=j) break;
        temp=A[i]; A[i]=A[j]; A[j]=temp;
    }
    temp=A[i]; A[i]=A[r]; A[r]=temp; // vratim pivot kam patri
    return i;
}

void quicksort(int A[], int l, int r){
if (l>=r) return;

// rozdelenie
int pivot=divide(A, l, r);

// reukrzivne volanie na mensie a vacsie prvky
quicksort(A,l,pivot-1);
quicksort(A,pivot+1,r);

//zlucenie nebude treba - su spravne za sebou
}
```

BubleSort

```
void swap(int &x, int &y) {
    /* Vymeň hodnoty premenných x a y. */
    int tmp = x;
    x = y;
    y = tmp;
}

void sort(int a[], int n) {
    /* usporiadaj prvky v poli a od najmenšieho po najväčší */

    bool hotovo = false;
    while (!hotovo) {
        bool vymenil = false;
        /* porovnávaj všetky dvojice susedov, vymeň ak menší za väčším */
        for (int i = 1; i < n; i++) {
            if (a[i] < a[i - 1]) {
                swap(a[i - 1], a[i]);
                vymenil = true;
            }
        }
        /* ak sme žiadnu dvojicu nevymenili, môžeme skončiť. */
        if (!vymenil) {
            hotovo = true;
        }
    }
}
```

Triedenie výberom (selection sort, max sort)

```
int maxIndex(int a[], int n) {
    /* vráť index, na ktorom je najväčší prvok z prvkov a[0]...a[n-1] */
    int index = 0;
    for(int i=1; i<n; i++) {
        if(a[i]>a[index]) {
            index = i;
        }
        /* invariant: a[j]<=a[index] pre vsetky j=0,...,i*/
    }
    return index;
}

void sort(int a[], int n) {
    /* usporiadaj prvky v poli a od najmenšieho po najväčší */

    for(int kam=n-1; kam>=1; kam--) {
        /* invariant: a[kam+1]...a[n-1] sú utriedené
         * a pre každé i,j také že 0<=i<=kam, kam<j<n platí a[i]<=a[j] */
        int index = maxIndex(a, kam+1);
        swap(a[index], a[kam]);
    }
}
```

Insertion Sort

```
void sort(int a[], int n) {
    /* usporiadaj prvky v poli a od najmenšieho po najväčší */

    for (int i = 1; i < n; i++) {
        int prvok = a[i];
        int kam = i;
        while (kam > 0 && a[kam - 1] > prvok) {
            a[kam] = a[kam - 1];
            kam--;
        }
        a[kam] = prvok;
    }
}
```

Sudoku

```
#include <iostream>
using namespace std;

void vypis(int **a) {
    /* vypis riesenie sudoku */
    for (int i = 0; i < 9; i++) {
        for (int j = 0; j < 9; j++) {
            cout << " " << a[i][j];
        }
        cout << endl;
    }
    cout << endl;
}

void najdiVolne(int **a, int &riadok, int &stlpec) {
    /* najdi volne policko na ploche a uloz jeho suradnice
     * do premennych riadpk a stlpec. Ak nie je, uloz -1. */
    for (int i = 0; i < 9; i++) {
        for (int j = 0; j < 9; j++) {
            if (a[i][j] == 0) {
                riadok = i;
                stlpec = j;
                return;
            }
        }
    }
    riadok = -1;
    stlpec = -1;
}

bool moze(int **a, int riadok, int stlpec, int hodnota) {
    /* Mozeme ulozit danu hodnotu na dane policko?
     * Da sa to, ak riadok, stlpec, ani stvorec nema
     * tuto hodnotu este pouzitu. */
    for (int i = 0; i < 9; i++) {
        if (a[riadok][i] == hodnota) return false;
        if (a[i][stlpec] == hodnota) return false;
    }
    /* lavy horny roh stvorca */
    int riadok1 = riadok - riadok % 3;
    int stlpec1 = stlpec - stlpec % 3;
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            if (a[riadok1 + i][stlpec1 + j] == hodnota) return false;
        }
    }
    return true;
}

int generuj(int **a) {
    /* mame ciastocne vyplnenu plochu sudoku,
     * chceme najst vsetky moznosti, ako ho dovyplnat
     * a vratit ich pocet.  */

    /* najdeme volne policko */
    int riadok, stlpec;
    najdiVolne(a, riadok, stlpec);

    /* ak uz ziadne nie je, vypiseme riesenie */
    if (riadok < 0) {
        vypis(a);
        return 1;
    } else {
        /* Do volneho policka dosadzujeme cifry 1..9,
         * volame rekurzivne vyplnanie ostatnych policok. */
        int pocet = 0; /* pocet rieseni */
        for (int x = 1; x <= 9; x++) {
            if (moze(a, riadok, stlpec, x)) {
                a[riadok][stlpec] = x;
                pocet += generuj(a);
                a[riadok][stlpec] = 0;
            }
        }
        return pocet;
    }

}

int main(void) {
    /* alokujeme a nacitame 2D maticu so vstupom */
    int **a = new int *[9];
    for (int i = 0; i < 9; i++) {
        a[i] = new int[9];
    }
    for (int i = 0; i < 9; i++) {
        for (int j = 0; j < 9; j++) {
            cin >> a[i][j];
        }
    }
    /* rekurzivne prehladavanie s navratom */
    int pocet = generuj(a);

    /* vypis riesenia */
    cout << "Pocet rieseni: " << pocet << endl;
    /* este by sme mali odalokovat polia v sudoku */
}
```

Vyfarbovanie

```
void vyfarbi(int **a, int n, int m, int riadok, int stlpec, int farba) {
    /* prefarbi suvislu jednofarebnu oblast obsahujucu
     * a[riadok][stlpec] na farbu farba */

    if (a[riadok][stlpec] != farba) {
        int stara_farba = a[riadok][stlpec];
        a[riadok][stlpec] = farba;
        if (riadok > 0 && a[riadok - 1][stlpec] == stara_farba) {
            vyfarbi(a, n, m, riadok - 1, stlpec, farba);
        }
        if (riadok + 1 < n && a[riadok + 1][stlpec] == stara_farba) {
            vyfarbi(a, n, m, riadok + 1, stlpec, farba);
        }
        if (stlpec > 0 && a[riadok][stlpec - 1] == stara_farba) {
            vyfarbi(a, n, m, riadok, stlpec - 1, farba);
        }
        if (stlpec + 1 < m && a[riadok][stlpec + 1] == stara_farba) {
            vyfarbi(a, n, m, riadok, stlpec + 1, farba);
        }
    }
}
```

Zoznam

```
void insertZac(zoznam &z, int d){
    item* p=new item;   // vytvoríme nový prvok
    p->data=d;          // naplníme dáta
    p->next=z.zaciatok; // prvok bude prvým prvkom zoznamu (ukazuje na doterajší začiatok)
    z.zaciatok=p;       // tento prvok je novým začiatkom
}
int deleteZac(zoznam &z){
     if (z.zaciatok==NULL) return ERR; // teda niečo o čom vieme, že je chyba

     item* p=(z.zaciatok->next); // prvok, ktorý bude ďalej začiatkom
     int d=z.zaciatok->data;     // to čo máme vrátiť
     delete z.zaciatok;          // uvoľnenie pamäte
     z.zaciatok=p;               // nový začiatok
     return d;
}
void vypis(zoznam z){
    item* x=z.zaciatok;
    while (x!=NULL){
        cout<<x->data<<" ";
        x=x->next;
    }
}
void remove(zoznam &z, int data) {
    // prvok sa v zozname nenachádza resp. je zoznam prázdny
    if (!z.zaciatok || z.zaciatok->data > data) return;

    // vymazávam prvý prvok
    if (data == z.zaciatok->data) {
        item* to_del = z.zaciatok;
        z.zaciatok = z.zaciatok->next;
        delete to_del;
        return;
    }

    //ostatné prípady
    item* prev = z.zaciatok;
    while (prev->next && prev->next->data < data) prev = prev->next;
    if (prev->next && prev->next->data == data) {
        item* to_del = prev->next;
        prev->next = prev->next->next;
        delete to_del;
    }
}
int deleteZac(zoznam &z){
     item* p=(z.zaciatok->next);
     int d=z.zaciatok->data;
     delete z.zaciatok;
     z.zaciatok=p;
     if (p!=NULL) p->prev=NULL;
     else z.koniec=NULL;
     return d;
}
```

Rad, fronta (queue)

```
/* inicializuje prázdny rad */
void init(queue &q);

/* zisti, ci je rad prazdny */
bool isEmpty(queue &q);

/* prida prvok s hodnotou item na koniec radu */
void enqueue(queue &q, dataType data);

/* odoberie prvok zo začiatku radu a vráti jeho hodnotu */
```

```
dataType dequeue(queue &q);

/* vrati prvok zo zaciatku radu, ale necha ho
v rade */
dataType peek(queue &q);
```

Zásobník (stack)

```
/* inicializuje prázdny zásobník */
void init(stack &s);

/* zisti, ci je zasobnik prazdny */
bool isEmpty(stack &s);

/* prida prvok s hodnotou item na vrch
zasobnika */
void push(stack &s, dataType data);

/* odoberie prvok z vrchu zásobníka a vráti
jeho hodnotu */
dataType pop(stack &q);
```

Postfix

Okrem infixovej notácie se môžeme stretnúť s ďalšími formami zápisu aritmetických výrazov. A to s prefixovou a postfixovou formou.
Postfixová notácia (obrátená poľská notácia) má v zápise výrazu operátor až po oboch operandoch.
Výraz (65 – 3*5)/(2 + 3) by teda v postfixovej forme mal notáciu 65 3 5 * - 2 3 + /
V prefixovej notácii je situácia opačná ako pri postfixovej notácii. Každý operátor stojí pred dvoma operandami, ku ktorým prislúcha.
Výraz (65 – 3*5)/(2 + 3) by v prefixovej forme bol / - 65 * 3 5 + 2 3
Postfixová a prefixová notácia sú na prvý pohľad pre človeka neprehľadné, ale ako uvidíme, dajú sa oveľa jednoduchšie vyhodnocovať. Okrem toho vidíme ešte jednu výhodu - nepotrebujú zátvorky.

Stromy

```
node * createOp(char op, node *left, node
*right) {
    /* vytvori novy vrchol stromu s operatorom
op
     * a do jeho laveho a praveho podvyrazu
ulozi
     * smerniky left a right. */
    node *v = new node;
    v->left = left;
    v->right = right;
    v->op = op;
    return v;
}

node * createNum(double val) {
    /* Vytvori novy vrchol stromu s danou
hodnotou,
     * lavy a pravy podvyraz bude prazdny, op
bude medzera */
    node *v = new node;
    v->left = NULL;
    v->right = NULL;
    v->op = ' ';
    v->val = val;
    return v;
}
//vyhodnocovanie vyrazu
double evaluate(node *v) {
    /* vyhodnoti vyraz dany stromom s korenom
vo vrchole v */
    assert(v != NULL);

    /* ak je operator medzera, vratime
jednoducho hodnotu */
    if (v->op == ' ') {
        return v->val;
    }

    /* rekurzivne vyhodnotime lavy a pravy
podvyraz */
    double valLeft = evaluate(v->left);
    double valRight = evaluate(v->right);

    /* Hodnotu laveho a praveho podvyrazu
spojime podla typu operatora
     * a vratime. Prikaz break netreba,
pouzivame return. */
    switch (v->op) {
        case '+': return valLeft + valRight;
        case '-': return valLeft - valRight;
        case '*': return valLeft * valRight;
        case '/': return valLeft / valRight;
        default: assert(0);
    }

void preorder(node *v) {
    if (v == NULL) return;
    print(v->data);
    preorder(v->left);
    preorder(v->right);
}

 int height(node *v) {
    /* Vrat vysku stromu s korenom v.
     * Ak v je NULL, vrati -1. */
    if (v == NULL) return -1;
    int left = height(v->left);
    int right = height(v->right);
    /* vrat max(left, right)+1 */
    if (left >= right) return left + 1;
    else return right + 1;
}
HLADANIE
node * createLeaf(keyType key, dataType data,
node * parent) {
    /* vytvor novy vrchol s danymi hodnotami,
obe deti nastav na NULL */
    node *v = new node;
    v->key = key;
    v->data = data;
    v->left = NULL;
    v->right = NULL;
    v->parent = parent;
    return v;
}

void findNode(node *root, keyType key, node
*&v, node *&parent) {
    /* Do v uloz smernik na vrchol s klucom
key alebo NULL ak neexistuje.
     * Do parent uloz otca v, NULL ak
neexistuje a ak key nie je v strome
     * tak smernik na vrchol, ktory by mal byt
otcom pre vrchol
     * s hodnotou key.*/
    parent = NULL;
    v = root;
    while (v != NULL && v->key != key) {
        parent = v;
        if (key < v->key) {
            v = v->left;
        } else {
            v = v->right;
        }
    }
}

void insert(dictionary &d, keyType key,
dataType data) {
    /* Do slovnika d vlozi kluc key a jeho
data.
     * Predpokladame, ze takyto kluc este v
slovniku nie je. */
    if (d.root == NULL) {
        /* prazdny strom - treba vytvorit
koren */
        d.root = createLeaf(key, data, NULL);
    } else {
        node *v;
        node *parent;
        findNode(d.root, key, v, parent);
        /* parent je teraz vrchol, ktoreho syn
ma byt novy vrchol */
        assert(v == NULL && parent != NULL);
        /* zisti, ci mame byt lave alebo prave
dieta otca */
        if (key < parent->key) {
            assert(parent->left == NULL);
            parent->left = createLeaf(key,
data, parent);
        } else {
            assert(parent->right == NULL);
            parent->right = createLeaf(key,
data, parent);
        }
    }
}
```

Trie

```
struct node {
    /* vrchol stromu  */
char letter;  /* pismeno vo vrchole alebo '?'
node * dot;   /* kod predlzeny o bodku */
node * dash;  /* kod predlzeny o ciarku */
};
void addCode(char letter, string code, node
*root) {
  /* Do stromu s koreňom root pridá kód
uložený v reťazci code
   * ktorý zodpovedá písmenu letter. */
    for(int i=0; i< code.length(); i++){
        if(code[i]=='.'){
            if(root->dot==NULL){
               root->dot= new node;

               root->dot->dot=  NULL;
               root->dot->dash= NULL;
               root->dot->letter='?';
               root=root->dot;

            }else{
                root=root->dot;
            }
        }else if(code[i]=='-'){
            if(root->dash==NULL){
               root->dash= new node;

               root->dash->dot=  NULL;
               root->dash->dash= NULL;
               root->dash->letter='?';
               root=root->dash;

            }else{
                root=root->dash;
            }
        }
    }
    root->letter=letter;
}
char findLetter(string message, int &pos, node
*root) {
  /* V strome s koreňom root nájde písmeno
zodpovedajúce kódu,
   * ktorý v texte message začína na pozícii
pos.
   * Toto písmeno vráti ako výsledok funkcie.
   * Ak takéto písmeno neexistuje, vráti znak
'?'.
   * Pozíciu pos posunie na najbližšiu medzeru
za kódom alebo za
   * posledné písmeno textu, ak za kódom je už
koniec textu. */
    while(pos<message.length())
    {
        if(message[pos]==' ') break;

        if(message[pos]=='.'){
            if(root->dot==NULL){
//pokial neexistuje vrchol , znamena to ze
user zadal blby kod, teda f-cia vracia '?'
//predtym ale musi preiterovat az na koniec
message-u
while (pos<message.length()) {
if (message[pos]==' ') break;
pos++;
 }  return('?');
            } else {
//inak je vsetko v poriadku a posunie sa po
strome
root=root->dot;
 pos++;
 }
//to iste pre dash
        }else if(message[pos]=='-'){
          if(root->dash==NULL) {

while (pos<message.length()) {
if (message[pos]==' ') break;
pos++;
}
return('?');} else {
root=root->dash;
pos++;
          }

        }

    }
//ak sa mu podari dostat az sem, znamena to ze
kod v message precital cely a
//pravdepodobne sa aj rovna nejakemu pismenu
(ak nie tak v letter bude '?'
//ktory aj tak tymto sposobom vypise
    return(root->letter);

}

//bonus(work in progress)


string najdi(node *root){  //zmenene z void na
string, vracias out
    string code, out;
    getline(cin,code);
    int j=0;
    while(j<code.length()){
    out = out + findLetter(code, j, root);
    j++;
    }
    return out;
}
 void nacitaj(node *root){

      ifstream fin("morse.txt"); // input
      char c,a;
      string code;
        while(!fin.eof()) {
            fin>>a>>code;
            addCode(a, code, root);

        }
}
int main() {
    node *p = new node;
    p->dash=NULL;
    p->dot=NULL;
    nacitaj(p);
    cout<<najdi(p);
 return 0;
}
```

Subory

```
ifstream fin("VSTUP.txt"); // input
char c;

while(!fin.eof()) {
  fin>>c;
  cout<<c;
}
```

SimpleDraw

```
#include "../SimpleDraw.h"

int main(void) {
    int width = 100;
    int n=6;
    /* Vytvor obrázok s rozmermi 300x400
pixelov */
    SimpleDraw window(300, 300);
    /* Nakresli obdĺžnik */
    window.drawRectangle(100, 100, width,
width);
    /* Nakresli čiaru */
    window.drawLine(100, 100, 100 + width, 100
+ width);
    /* Korytnačky */
    Turtle turtle(window, (300 - width) / 2,
300 - 1, 0);
    for (int i = 0; i < n; i++) {
        turtle.forward(length);
        turtle.turnLeft(360.0 / n);
    }
    /* Ulož obrázok do súboru s príponou png.
    window.savePng("obrazok.png");
    /* Zobraz na obrazovke a čakaj, kým
užívateľ stlačí Exit,
       potom zavri okno. */
    window.showAndClose();
```

Premenne

```
int A[4]={3, 6, 8, 10}; //spravne
int B[4];               //spravne
B[4]={3, 6, 8, 10};     //nespravne
B[0]=3; B[1]=6; B[2]=8; B[3]=10;
char C[100] = "Edo";
char ch = 'l';
char D[100] = "pes";
char E[100];
D[0] = ch;     // priradíme do jedného prvku
reťazca premmennú typu char. Výsledkom je
'les'.
D[0] = 'v';    // priradíme do jedného prvku
reťazca konštantný znak. Výsledkom je 'ves'.
D[0] = C[1];   // priradíme do jedného prvku
reťazca prvok iného reťazca. Výsledkom je
'des'.
E[0]='a'; E[1]='h'; E[2]='o'; E[3]='j';
E[4]=0; //alebo E[4]='\0'; reťazec musí končiť
0.
```

Premenne

```
int i    // „klasická“ celočíselná premenná
int *p_i // ukazovateľ na celočíselnú premennú
p_i=&i;          // spravne
p_i = &(i + 3); // zle i+3 nie je premenna
p_i=&15;         //zle konstanta nema adresu
i=*p_i;          // spravne ak p_i bol
inicializovany

int * cislo = new int;
*cislo = 50;
..
delete cislo;
int a[4];
int *b = a;  /* a,b su teraz takmer rovnocenne
premenne */
int *A;
A = new int[n];..
delete[] A
int **a;
a = new int *[n];
for (int i = 0; i < n; i++) a[i] = new int[m];
..
for (int i = 0; i < n; i++) delete[] a[i];
delete[] a;
```