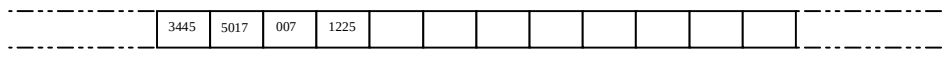
שימוש בקבצים חיצוניים

הזיכרון הפרמננטי

עד כה התרכזנו במימוש של מבנה נתונים שונים. כאשר מדברים על מבני נתונים, מדברים על ארגון אינפורמציה בזיכרון המחשב. על הזיכרון אפשר להסתכל כעל רצף של תאים בהם ניתן להכניס ערכים. אחד המאפיינים החשובים של הזיכרון הוא ששמירת האינפורמציה בו היא זמנית, נמשכת כל עד התוכנית רצה במחשב. פעמים רבות אנחנו רוצים לאכסן את האינפורמציה באופן קבוע, אכסון שישמר גם לאחר כיבוי המחשב. לשם כך אנו משתמשים באמצעי אכסון פרמננטיים – דיסקטים, דיסק קשיח, cd וכ׳.
לאמצעי האחסון הפרמננטי נתייחס בדומה לזיכרון, כרצף של תאים בהם ניתן להכניס ערכים:



יחידת אינפורמציה בסיסית בזיכרון הפרמננטי נקראת **קובץ**. לקובץ ניתן להתייחס כסדרה של מספרים המקודדים אינפורמציה מסוימת. מהות אותה אינפורמציה מקודדת נקבעת לפי סוג הקובץ. הסימון המקובל לסוג הקובץ הוא ה – extension שלו : סידרת האותיות המופיעות לאחר הנקודה בשם הקובץ. האינפורמציה המקודדת כמספרים בקובץ יכולה להיות מסוגים שונים:
text, תמונה, קול, בסיסי נתונים וכ׳. בדומה למשתנים, אפשר לחשוב על סוג הקובץ **כטיפוס** שלו. גם במשתנים, הטיפוס הוא הקובע את **האינטרפטציה** של הערך המספרי המוכל.

אנחנו נכתוב קוד ב – C++ המשתמש בזיכרון הפרמננטי בכדי לשמור ולאחזר אינפורמציה שנרצה לשמור באופן קבוע.

פתיחה של קבצים לקריאה ולכתיבה

בכדי להשתמש בשירותים המאפשרים שימוש בקבצים עלינו לכלל את קובץ ה – header הבא :

```
#include <fstream>
```

המחלקה המייצגת `stream` לקובץ כתיבה היא : `ofstream` . הקוד הבא מדגים שימוש במחלקה זו בכדי לכתוב אינפורמציית `text` לקובץ ב – `directory` הנוכחית:

```
#include <fstream>
int main() {
        std::ofstream outputFile;
        outputFile.open("out-file.txt");
        if(!outputFile) {
                std::cerr << “can’t open out-file.txt\n”;
                std::exit(1);
        }
        outputFile << "here is an example of a text file..\n"
                << " the way we do it is similar to the way we send text \n"
                << " to the standard output stream" << std::endl;
}
```

אם קיימת בעיה לפתוח את הקובץ האופרטור ׳!׳ יחזיר ערך true.

את האינפורמציה ששמרנו בקובץ ”out-file.txt“ ניתן לאחזר באופן הבא :

```
#include <fstream>
int main() {
        std::ifstream fl;
        fl.open("out-file.txt");
```

```
        if(!fl) {
                std::cerr << "problem opening input file\n";
                exit(1);
        }
        char buf[20]; // max word size
        while(!fl.eof()) { // while not end of file
                fl >> buf; // read a word (untill a separator)
                std::cout << buf << " ";
        }
}
```

אפשרות נוספת לקריאה היא קריאה תו-תו בעזרת `get`:

```
        char c;
        while(fl.get(c)) {
                std::cout.put(c);
        }
```

קביעת מיקום הקובץ במערכת הקבצים

ניתן לקבוע את מיקום הקובץ במערכת הקבצים ע"י מחרוזת המכילה path יחסי או אבסולוטי:

```
file.open("/home/me/stam.data");
file.open("../../anotherDir/stam.txt");
```

שמירת אובייקטים בקובץ

ניתן לשמור אובייקטים בזיכרון הפרמננטי בצורה נוחה כאשר מגדירים פונקציות מתאימות עבור `ostream` ו- `istream`:

```
#include <fstream>

class Point {
public:
        double _x,_y;
        Point(double x=0, double y=0)
                : _x(x),_y(y) {}
};

std::ostream& operator <<(std::ostream& os, const Point& p) {
        os << '(' << p._x << ',' << p._y << ')';
        return os;
}

std::istream& operator >>( std::istream& is, Point& p) {
        char c;
        is >> c >> p._x >> c >> p._y >> c;
        return is;
}


int main() {
        Point p;
        std::cout << "please enter a point\n";
        std::cin >> p;
```

```
        std::ofstream out("points");
        out << p;
        p._x = 17;
        out << p;
        out.close(); // close the file
        std::ifstream in("points");
        in >> p; std::cout << p;
        in >> p; std::cout << p;
}
```

המחלקה `ofstream` יורשת מהמחלקה `ostream` והמחלקה `ifstream` יורשת מ – `istream`. מספיק שנסביר איך להעביר אובייקט ל – `stream` כדי לדעת איך להעביר אותו ל – `stream` של קובץ.
ניתן להשתמש באותו רעיון כדי לשמור ולשחזר אובייקטים מורכבים יותר:

```
class Triangle {
public:
        Point _p1,_p2,_p3;
};

std::ostream& operator <<( std::ostream& os, const Triangle& tr) {
        os << '[' << tr._p1 << ','<< tr._p2 << ','
                << tr._p3 << ']';
        return os;
}

std::istream& operator >>( std::istream& is, Triangle& tr) {
        char c;
        is >> c >> tr._p1 >> c >> tr._p2 >> c
                >> tr._p3 >> c;
        return is;
}
int main() {
        Triangle tr;
        std::cout << "please enter a triangle\n";
        std::cin >> tr;
        std::ofstream out("triangle");
        out << tr;
        out.close();
        std::ifstream in("triangle");
        Triangle t2;
        in >> t2;  cout << t2;
}
```

באותו אופן נוכל לכתוב פונקציות המעבירות כל מבנה נתונים ל – `stream` ומתוך `stream`. לתהליך זה קוראים `serialization`, כלומר קידוד האובייקט למהות סידרתית. רצוי שהקידוד (הכתיבה ל – `stream`) והפיענוח (הקריאה ממנו) יהיו לפי אותו הפורמט כך שיתאפשר פיענוח של אותה אינפורמציה שקודדה.

דוגמה: כתיבה וקריאה של עץ משולשים

הקוד הבא מדגים עץ בינארי ממויין של משולשים הניכתב לקובץ וניקרא מימנו. שימו לב בעיקר לפונקציה get האחראית על הקריאה:

```
#include <fstream>

class Point {
public:
	double _x,_y;
	Point(double x=0, double y=0)
		: _x(x),_y(y) {}
};

std::ostream& operator <<( std::ostream& os, const Point& p) {
	os << '(' << p._x << ',' << p._y << ')';
	return os;
}

std::istream& operator >>( std::istream& is, Point& p) {
	char c;
	is >> c >> p._x >> c >> p._y >> c;
	return is;
}
class Triangle {
public:
	Point _p1,_p2,_p3;
	double val() { return _p1._x+_p2._x*2+_p3._x; }
	Triangle() {}
	Triangle(Point& p1, Point& p2, Point& p3):
		_p1(p1), _p2(p2), _p3(p3) {}
};

std::ostream& operator <<( std::ostream& os, const Triangle& tr) {
	os << '[' << tr._p1 << ','<< tr._p2 << ','
		<< tr._p3 << ']';
	return os;
}

std::istream& operator >>( std::istream& is, Triangle& tr) {
	char c;
	is >> c >> tr._p1 >> c >> tr._p2 >> c
		>> tr._p3 >> c;
	return is;
}

class Node {
public:
	Triangle _data;
	Node *_ls,*_rs;
	Node(const Triangle& data, Node* ls = NULL, Node* rs = NULL);
	Node(Node* ls = NULL, Node* rs = NULL);
```

```
};
Node::Node(const Triangle& data, Node* ls, Node* rs):
        _data(data),_ls(ls),_rs(rs) {}
Node::Node(Node* ls, Node* rs):
        _ls(ls),_rs(rs) {}
// could have just the first ctor:
// Node(const Triangle& data=Triangle(), Node* ls = NULL, Node* rs = NULL);

class Tree {
        friend ofstream& operator <<(ofstream& os, const Tree& tree);
        friend ostream& operator <<(ostream& os, const Tree& tree);
        friend istream& operator >>(istream& is, Tree& tree);
        static void print(ostream& os, Node* p);
        static void get(istream& is, Node** p);
        Node* _root;
public:
        Tree();
        void insert(Triangle& data);
};

Tree::Tree() : _root(NULL) {}

std::ostream& operator <<( std::ostream& os, const Tree& tree) {
        os << "a tree:\n";
        Tree::print(os,tree._root);
        return os;
}

std::ofstream& operator <<( std::ofstream& os, const Tree& tree) {
        Tree::print(os,tree._root);
        return os;
}

void Tree::insert(Triangle& data) {
        Node** p = &_root;
        while(*p) {
                if(data.val()<((*p)->_data).val())
                        p = &((*p)->_ls);
                else
                        p = &((*p)->_rs);
        }
        *p = new Node(data);
}

void Tree::print(ostream& os, Node* p) {
        if(!p) {
                os<<'n';
                return;
        }
        os << '{';
```

```
        print(os,p->_ls);
        os << ',' << p->_data <<',';
        print(os,p->_rs);
        os<<'}';
}

std::istream& operator >>( std::istream& is, Tree& tree) {
        Tree::get(is,&(tree._root));
        return is;
}

void Tree::get(std::istream& is, Node** p) {
        char c;
        is>>c;
        if(c == 'n') {
                *p = NULL;
                return;
        }
        *p = new Node;
        // the '{' was already read
        get(is,&((*p)->_ls));
        is >> c >> (*p)->_data >> c;
        get(is,&((*p)->_rs));
        is.get(c); // '}'
}

int main() {
        Tree tree;
        Point* array[3];
        array[0] = new Point(1,1);
        array[1] = new Point(2,2);
        array[2] = new Point(3,3);
        Triangle* arrT[6];
        int ix = 0;
        for(int i=0; i<3; i++)
                for(int j=(i+1)%3; j!=i; j=(j+1)%3)
                        arrT[ix++] = new Triangle(*array[i],*array[j],*array[3-i-j]);
        for(int t=0; t<6; t++)
                tree.insert(*arrT[t]);

        std::ofstream of("trees.txt");
        std::cout << tree << endl;
        of << tree;
        // delete ...
        of.close();
        Tree tree1;
        std::ifstream in("trees.txt");
        in >> tree1;
        cout << tree1;
}
```

בכל הדוגמאות שראינו, השתמשנו בקבצי טקסט. הקידוד שלנו היה יכל להיות חסכני יותר אם היינו משתמשים בקבצים של מספרים ומקפידים יותר על קומפקטיות הקידוד.

ישנה אפשרות ליצור קובץ הנועד גם לקריאה וגם לכתיבה. כמו כן ישנן אפשרויות להזזה של מקום הקריאה ומקום הכתיבה בקובץ. פרטים ניתן למצוא C++ primer, p: 1097 או ב – Stroustrup, p:637.