Informatika 3
9. előadás

Tóth Dávid

Budapesti Műszaki és Gazdaságtudományi Egyetem

2025.05.05.

Karakterláncok

- Korábban C stílusú karakterláncokkal foglalkoztunk.
- Ezek `char` típusú elemeket tartalmazó tömbök, melyek a `'\0'` karakterrel végződnek.
- Korábban: az `strlen` függvény (`cstring` fejállomány) megadja a karakterlánc hosszát a záró `'\0'` karakter nélkül.
- További hasznos függvények a karakterláncok kezelésére:
  - `strcpy_s`: C stringek másolása
  - `strcat_s`: C stringek összefűzése
  - `strcmp`: C stringek lexikografikus rendezése (0-t ad vissza, ha egyenlők, negatív értéket, ha az első string kisebb, mint a második, egyébként pozitív értéket)
- Karakterláncok konvertálása más típussá (`cstdlib`):
  - `atoi`, `atol`, `atoll`: C string -> `int`, `long`, `long long`
  - `atof`: C string -> `double`

A string osztály

- C stringek helyett sokszor kényelmesebb a `string` osztállyal dolgozni, amely a `string` fejállományban található.
- Néhány lehetőség a `string` inicializálására:
  - `string s;` - üres `string`
  - megadhatunk egy C stringet kezdeti értékként:

```
const char* word = "asd";
string s1(word);
string s2("Felix Klein");
```

  - `char` típust nem adhatunk meg konstruktor paramétereként:

```
string s1('c'); //hiba
```

  - DE: a következő kód működik:

```
string s1;
s1 = 'c'; //ez OK
```

  - szintén jó a `string(size_t nreps, char ch)` deklaráció, `nreps` számú `ch` karaktert ír a `string`-be, pl.:

```
string s1(10,'o');
```

A string osztály

- További lehetőségek a string inicializálására:
  - string másolása:

```
string s1("asd");
string s2(s1);  //s1 másolása s2-be
```

  - string másolása adott pozíciótól:

```
string s1("abcdefg");
string s2(s1,2); //s1 másolása a 2 indexű karaktertő
//s2 tartalma "cdefg"
```

  - string adott hosszúságú részének másolása adott pozíciótól:

```
string s1("abcdefg");
string s2(s1,2,4);
//s2 tartalma "cdef"
```

Operációk a string osztályon

- A + operátor konkatenálja a string-eket, az operandus lehet char is:

```
string s1("Arrivederci");
string s2("palotapincsi");
string s3 = s1 + " " + s2 + '!';
//az s3 tartalma "Arrivederci palotapincsi!"
```

- A += operátorral hozzáfűzhetünk egy string-hez egy másikat.
- Az insert tagfüggvény beszúr egy string-et egy másik string megadott pozíciójától kezdve.

```
string s1("aa");
s1.insert(1,"bb");
// az s1[1] karakter elé fűzzük a "bb" stringet
```

- Az insert első paramétere a pozíció, a második egy C string vagy string.

Operációk a string osztályon

- Az erase tagfüggvénnyel törölhetünk a string-ből:

```
string s = "abcdefghijk";
s.erase(6);
//a 6 indexű karaktertől kezdve törli a string-et
//s tartalma "abcdef";
s.erase(1,3);
//az 1 indexű karaktertől kezdve töröl 3 karaktert
//s tartalma "aef"
```

- Az replace tagfüggvénnyel lecserélhetünk karaktereket a string-ben:

```
string s = "abcdefghijk";
s.replace(3,5,"...");
//a 3 indexű karaktertől 5 karaktert cserél "..."-ra
//s tartalma ezután "abc...ijk"
//a 3. paraméter lehet char* vagy string
```

Operációk a string osztályon

- A substr tagfüggvény a string egy részletét adja vissza, a visszatérési érték is string.

```
string s1 = "abcdefghijk";
string s2 = s.substr(3);
//az s1 3 indexű karakterétől
//az s1 végéig tartó karaktersorozatot adja vissza
s2 = s.substr(3,4);
//az s1 3 indexű karakterétől számított
//4 karakterből álló karaktersorozatot adja vissza
```

- Az s string i indexű karakterének elérése: s[i] vagy s.at(i)

Rendezés a `string` osztályon

- Két `string` összehasonlítható az ==, !=, <, >, <=, és >= operátorokkal.
- A rendezés lexikografikus, de a karakterek kódján alapul, így pl. a nagy betűk megelőzik a kis betűket, a space karakter megelőzi a betűket stb.

```
string words[] = { "ABBA", "aaa", "Bab", " zebra" };
sort(words, words + 4);
//a rendezett sorozat " zebra", "ABBA", "Bab", "aaa"
```

Iterátorok

- Iterátorokkal is bejárhatjuk a string karaktereit:

```
string s = "abba";
for(string::iterator it = s.begin();
                        it != s.end(); ++it) {
    cout << (*it)++;
}
```

- A fenti bejárás tartomány alapú for ciklussal:

```
string s = "abba";
for(char& c : s){
    cout << c++;
}
```

- Az utóbbi kódban a `c` referencia végigfut az `s` `string` elemein.
- Mit ír ki a kód, és mi lesz az `s` tartalma a futás után?

Iterátorok

- Az iterátorokat használhatjuk az `erase` tagfüggvénnyel való törlésnél a törlendő rész első ill. utolsó utáni karakterének megadására:

```
string s = "abba";
string::iterator it1 = s.begin()+1;
string::iterator it2 = s.end()-1;
s.erase(it1, it2);
```

- A fenti kód kitörli a másodiktól az utolsó előtti karakterig az összes karaktert.
- Ha csak egy paramétert adunk meg, akkor csak azt a karaktert törli.

Iterátorok

- Az `insert` tagfüggvénnyel egy karaktert szúrhatunk be az iterátor által mutatott karakter elé:

```
string s = "abba";
string::iterator it = s.begin()+1;
s.insert(it, 'c');
//char*-gal vagy string-gel nem működik
```

- A `replace` a két iterátor által mutatott karakterek közti rész cseréli le a harmadik paraméterként megadott stringre:

```
string s = "abba";
string::iterator it1 = s.begin()+1;
string::iterator it2 = s.end()-1;
s.replace(it1, it2, "BB");
```

Keresés `string`-ben

- Egy `string`-ben a `find` tagfüggvény segítségével kereshetünk egy adott karaktersorozatot.
- A függvény paramétere a keresett karaktersorozat, `char*` vagy `string` típusú.
- A visszatérési érték az első előfordulás indexe, ez `string::size_type` típusú (`int`-té konvertálható).
- Ha nem található a keresett string, akkor a visszatérési érték `string::npos`.

Keresés `string`-ben

- Második paraméterként megadhatjuk, hogy a `find` hányadik indextől keressen a stringben, így megkereshetjük az összes előfordulást:

```
string s = "Mississippi";
string subs = "issi";
string::size_type idx = s.find(subs);
int count_subs = 0;
while (idx != string::npos) {
    count_subs++;
    idx = s.find(subs, idx+1);
}
```

- Az `rfind` tagfüggvény visszaadja a karaktersorozat utolsó előfordulásának indexét, vagy `string::npos`-t, ha nem fordul elő, második paraméterként megadható az utolsó kezdőkarakterként figyelembe vett karakter indexe.

Keresés `string`-ben

- A `find_first_of` ill. `find_last_of` tagfüggvények megadják az első ill. utolsó indexet, ahol a paraméterként megadott karaktersorozat valamelyik karaktere előfordul.
- Ha egyik karakter sem fordul elő, akkor a visszatérési érték `string:npos`.
- Mi az `idx` értéke a következő kódban?

```
string s = "Mississippi";
string subs = "ips";
string::size_type idx = s.find_last_of(subs);
```

- Második paraméterként beállítható az első/utolsó karakter, ahonnan/ameddig keresünk.
- A `find_first_not_of` ill. `find_last_not_of` függvények argumentumában azon karaktereket adhatjuk meg, amelyektől különböző első/utolsó karakter indexét keressük (a kezdő pozíció ugyanúgy állítható, mint fent).

Konverzió

- Ha `char*`-ot `string`-gé szeretnénk konvertálni, ez az = operátorral lehetséges:

```
string s;
s = "abcd";
```

- Ezt a `string` deklarációjánál is megtehetjük:

```
string s("abcd");
```

- `string`-ből `const char*`-gá a `c_str` tagfüggvénnyel konvertálhatunk.
- `string`-ből `char*`-gá szeretnénk konvertálni, akkor a `c_str` által visszaadott pointer által mutatott tömb tartalmát át kell másolnunk egy másik tömbbe (ritka, hogy erre szükség van).

Paraméter default értéke

- A `string` osztály `find` tagfüggvényét meghívhatjuk többféleképpen:

```
string s = "Mississippi"
string::size_type idx1 = s.find("is");
string::size_type idx2 = s.find("is",2);
```

- Ehhez a `find` tagfüggvényt nem kell kétféle paraméterezéssel deklarálni, a második paraméternek megadható egy default érték (ez esetben 0).

```
string::size_type string::find(const char* s,
                               string::size_type pos = 0);
```

- Ha egy paraméternek van default értéke, az összes őt követő paraméternek kell default értéket adni:

```
int f(int a = 0, int b);
//hiba: a b-nek nincs default értéke
```

size_t és size_type

- A size_t egy típus, amit objektumok méretének leírására használnak.
- A size_type név tárolókban tárolt elemek számát leíró típus neve, ez a típus az std könyvtárban a size_t.
- Más könyvtárak tárolóinál ez változhat (pl. lehet int), így érdemes a container::size_type nevet használni size_t helyett.
- A size_t használatánál vigyázni kell, mert ez unsigned érték:

```
string s;
for(size_t i = 0; i < s.size() - 1; ++i){
    s[i] = 'c';
}
```

- Az s.size() - 1 értéke size_t típusú, mégpedig az unsigned size_t típus maximális értéke...

Szöveges file-ok kezelése

- Ha a programunk által generált adatot később használni szeretnénk (pl. tesztelni szeretnénk, hogy a véletlenszám-generátor értékei valóban úgy viselkednek-e, ahogy elvárjuk), akkor azokat file-okba is kiírhatjuk.
- A file-oknak két alapvető típusa a *szöveges* ill. *bináris* file-ok.
- Szöveges file-okban karaktersorozatokat tárolunk, míg bináris file-okban tetszőleges adatot.
- Most szöveges file-okkal foglalkozunk, ezek kezeléséhez az `fstream` fejállományra lesz szükségünk.

Szöveges file-ok kezelése

- A file-ok írásához az `std::ifstream` (input file stream), míg az olvasáshoz az `std::ofstream` (output file stream) objektumot használjuk.
- A deklarálásnál paraméterként megadjuk a file nevét:

```
std::ifstream f_in("input_file");
std::ofstream f_out("output_file");
```

- Előfordulhat, hogy a file-ok megnyitása nem sikerül, pl. olvasnánk egy nem létező file-t, nem tudunk írni egy file-ba, mert egy másik alkalmazás használja stb.
- Mindkét osztálynak van `fail()` ill. `good()` tagfüggvénye.
- A `fail()` igazat ad vissza, ha nem sikerül megnyitni a file-t írásra/olvasásra, és hamisat különben.
- A `good()` igazat ad vissza, ha sikerül megnyitni a file-t írásra/olvasásra, és hamisat különben.

Szöveges file-ok kezelése

- Ezen hibák lekezelése pl.:

```
if(f_in.fail()){
    std::cerr << "A file nem olvashato."  << endl;
    return 1;
}
```

- Ha nem létező file-t nyitunk meg írásra, akkor a file létrejön.
- Ha létező file-t nyitunk meg írásra, akkor a tartalma törlődik.
- Ha egy létező file-ba írni szeretnénk, de nem szeretnénk törölni a tartalmát, azt a következő deklarációval érhetjük el:

```
std::ofstream f_out(file neve, ios::app);
```

- Így megnyitva a file-t az író utasítások a file végére írnak.

Írás és olvasás

- Ha egy `ofstream` objektummal egy file-t sikerült megnyitni írásra, akkor `cout`-hoz hasonlóan a `<<` operátor segítségével írhatunk a file-ba:
- ```
using namespace std;
int main() {
    ofstream f_out("output_file");
    if(f_out.fail()) {
        cerr << "A file nem nyithato meg." << end;
        return 1;
    }
    for(int i = 1; i <= 10; ++i) {
        f_out << i << " ";
    }
    f_out << endl;
    return 0;
}
```
```

Írás és olvasás

- A >> operátorral olvashatunk egy `ifstream` objektum segítségével megnyitott file-ból.
- Olvashatunk `string`, `int`, `double`, `char`, stb. típusú változókba.
- Ha a >> operátorral való olvasás sikertelen (pl. a file végére érünk, az olvasott típusnak nem megfelelő formátumú a string), akkor az olvasási kísérlet után a `fail()` tagfüggvény igazat, vagy a `good()` tagfüggvény hamisat ad vissza.
- Ha egy sikertelen olvasás után tovább szeretnénk folytatni az olvasást, akkor a `clear()` tagfüggvényt kell előtte meghívni, különben a `fail()` tagfüggvény továbbra is igazat fog visszaadni minden kísérletnél.

Olvasás file-ból

- A >> operátorral `string` típusú változóba is olvashatunk a file-ból, ez a szóközöket figyelmen kívül hagyja, és mindig a következő karaktersorozat utáni első szóközig vagy sorzárásig olvas, a beolvasott `string`-be ezek nem kerülnek be.
- Ha karakterenként szeretnénk olvasni, akkor `char` típusú változóba olvassunk a `get` tagfüggvénnyel. (A >> operátor ebben az esetben is átugorja a szóközöket és sortöréseket.)
- A `string` fejállományban lévő `getline` függvénnyel beolvashatunk egy egész sort egy `string`-be:

```
ifstream f_in(filename);
string s;
getline(f_in,s);
```

- A file végét ezekben az esetekben is a `ifstream` `fail` tagfüggvényével detektálhatjuk.

Olvasás file-ból

- A >> operátorral számokat is olvashatunk (egész és lebegőpontos értékeket).
- A `fail` tagfüggvény rossz formátum esetén hamisat fog visszaadni az olvasási kísérlet után.
- Ezek után egy újabb olvasási kísérlethez meg kell hívni a `clear` tagfüggvényt.
- Ha a file-t nem használjuk tovább, a `close` tagfüggvénnyel zárjuk be.
- File-t megnyitni az `open` tagfüggvénnyel is tudunk.

String stream

- Megjegyezzük, hogy a `get`, `fail`, `clear` tagfüggvények a `cin` `istream` esetén is hasonlóképp használhatók (pl. rossz olvasási kísérlet detektálására), a >> operátor pedig úgy viselkedik, mint ahogy fent leírtuk.
- `string`-ből is készíthető stream az `istringstream` objektum segítségével (`sstream` fejállomány):

```
string s("abcdef");
istringstream is(s);
```

- Erre a fenti eszköztár ugyanúgy használható.
- Az `ostringstream` objektum hasonlóan működik, mint az `ostream` vagy az `ofstream` objektumok, de az eredményt az `str` tagfüggvénnyel `string` formájában nyerhetjük ki.