**Лабораторна робота №9**

з дисципліни «Алгоритмізація та програмування» Тема: "Робота з рядками на С ++"

МЕТА РОБОТИ

Вивчити теоретичний матеріал з основ роботи з низькорівневими рядками на С++ і документацію до класу string, а також алгоритми пошуку в рядку, а також реалізувати обробку рядків на C++ в середовищі Visual Studio.

ПОСТАНОВКА ЗАДАЧІ

Завдання 1.

1. Вивчити по документації метод стандартного класу string відповідно до варіанту (*див. табл.1*).
2. Визначити функцію, що виконує ті ж дії, що і вивчений метод класу string. Вихідний рядок передати першим параметром (масив символів). Для реалізації методу не використовувати функції обробки рядків зі стандартних бібліотек.
3. Викликати свій метод і метод string *аналогічно прикладам коду, наведеними в дод.А*. \*Перед викликом ввести з консолі один рядок і зберегти в масиві символів і змінній типу string.

Завдання 2.

1. Описати функцію, що обробляє рядок відповідно до *завдання з табл.2.* Для реалізації можна використовувати функції обробки рядків зі стандартних бібліотек
2. Описати функцію, яка перевіряє, чи задовольняє рядок умовам завдання.
3. \* Створити вихідний текстовий файл, що містить не менше 10 різних рядків.
4. Використовуючи функції 2.А і 2.В, обробити рядок / \* текстовий файл рядок за рядком. Додаткові дані ввести з консолі.
5. Отриманий результат записати у вихідний файл.

Звіт роботи:

// appending to string #include <iostream> #include <string>

int main ()

{

std :: string str;

std :: string str2 ="Writing";

std :: string str3 ="Print 10 and then 5 more";

// used in the same order as described above: str.append (str2); // "Writing"

str.append (str3,6,3); // "10" str.append ("Dots are cool", 5); // "dots" str.append ("Here:"); // "here:"

str.append (10u,'.'); // ". "

std :: cout << str << '\ N'; return 0;

}

Output:

Writing 10 dots here: .......... and then 5 more .....

// comparing apples with apples #include <iostream>

#include <string>

int main ()

{

std :: string str1 ("Green apple"); std :: string str2 ("Red apple");

if (Str1.compare (str2)! = 0)

std :: cout << str1 << "Is not" << str2 << '\ N';

if (Str1.compare (6,5,"Apple") == 0)

std :: cout << "Still," << str1 << "Is an apple \ n";

if (Str2.compare (str2.size () - 5,5,"Apple") == 0)

std :: cout << "And" << str2 << "Is also an apple \ n";

if (Str1.compare (6,5, str2,4,5) == 0)

std :: cout << "Therefore, both are apples \ n";

return 0;

}

Output:

green apple is not red apple still, green apple is an apple and red apple is also an apple therefore, both are apples

// string :: assign #include <iostream> #include <string>

int main ()

{

std :: string str;

std :: string base ="The quick brown fox jumps over a lazy dog.";

// used in the same order as described above: str.assign (base);

std :: cout << str << '\ N';

str.assign (base, 10,9);

std :: cout << str << '\ N';

// "brown fox"

str.assign ("Pangrams are cool", 7);

std :: cout << str << '\ N'; // "pangram"

str.assign ("C-string"); std :: cout << str << '\ N';

// "c-string"

str.assign (10,'\*');

std :: cout << str << '\ N';

// "\*\*\*\*\*\*\*\*\*\*"

str.assign <int> (10,0x2D); std :: cout << str << '\ N';

// " "

str.assign (base.begin () + 16, base.end () - 12);

std :: cout << str << '\ N'; // "fox jumps over"

return 0;

}

Output:

The quick brown fox jumps over a lazy dog. brown fox

pangram c-string

\*\*\*\*\*\*\*\*\*\*

fox jumps over

// inserting into a string #include <iostream> #include <string>

int main ()

{

std :: string str ="To be question"; std :: string str2 ="The";

std :: string str3 ="Or not to be"; std :: string :: iterator it;

// used in the same order as described above: str.insert (6, str2);

str.insert (6, str3,3,4); str.insert (10,"That is cool", 8); str.insert (10,"To be");

question

str.insert (15,1,':');

// to be (the) question

// to be (not) the question

// to be not (that is) the question

// to be not (to be) that is the

// to be not to be (:) that is the

question

it = str.insert (str.begin () + 5,','); // to be (,) not to be: that is the question

str.insert (str.end (), 3,'.'); // to be, not to be: that is the question (...)

str.insert (it + 2, str3.begin (), str3.begin () + 3); // (or)

std :: cout << str << '\ N'; return 0;

}

Output:

to be, or not to be: that is the question ...

// string :: erase #include <iostream> #include <string>

int main ()

{

std :: string str ("This is an example sentence."); std :: cout << str << '\ N';

// "This is an example sentence."

str.erase (10,8); // ^^^^^^^^ std :: cout << str << '\ N';

// "This is an sentence."

str.erase (str.begin () + 9); // ^ std :: cout << str << '\ N';

// "This is a sentence." str.erase (str.begin () + 5, str.end () - 9); // ^^^^^

std :: cout << str << '\ N';

// "This sentence."

return 0;

}

Output:

This sentence.

is

an example

This is an sentence. This is a sentence. This sentence.

// replacing in a string #include <iostream> #include <string>

int main ()

{

std :: string base ="This is a test string."; std :: string str2 ="N example";

std :: string str3 ="Sample phrase"; std :: string str4 ="Useful.";

// replace signatures used in the same order as described above:

// Using positions: 0123456789 \* 123456789 \* 12345

std :: string str = base; // "this is a test string." str.replace (9,5, str2); // "this is an example string." (1) str.replace (19,6, str3,7,6); // "this is an example phrase." (2) str.replace (8,10,"Just a"); // "this is just a phrase." (3) str.replace (8,6,"A shorty", 7); // "this is a short phrase." (4) str.replace (22,1,3,'!'); // "this is a short phrase !!!" (5)

// Using iterators: 0123456789 \* 123456789 \*

str.replace (str.begin (), str.end () - 3, str3); // "sample phrase !!!" (1)

str.replace (str.begin (), str.begin () + 6,"Replace"); // "replace phrase !!!" (3)

str.replace (str.begin () + 8, str.begin () + 14,"Is coolness", 7); // "replace is cool !!!" (4)

str.replace (str.begin () + 12, str.end () - 4,4,'O'); // "replace is cooool !!!" (5)

str.replace (str.begin () + 11, str.end (), str4.begin (), str4.end ());// "replace is useful." (6)

std :: cout << str << '\ N'; return 0;

}

Output:

replace is useful.

// swap strings #include <iostream> #include <string>

main ()

{

std :: string buyer ("Money"); std :: string seller ("Goods");

std :: cout << "Before the swap, buyer has" << buyer; std :: cout << "And seller has" << seller << '\ N';

seller.swap (buyer);

std :: cout << "After the swap, buyer has" << buyer; std :: cout << "And seller has" << seller << '\ N';

return 0;

}

Output:

Before the swap, buyer has money and seller has goods After the swap, buyer has goods and seller has money

// string :: copy #include <iostream> #include <string>

int main ()

{

char buffer [20];

std :: string str ("Test string ...");

std :: size\_t length = str.copy (buffer, 6,5); buffer [length] ='\ 0';

std :: cout << "Buffer contains:" << buffer << '\ N'; return 0;

}

Output:

buffer contains: string

// string :: find #include <iostream> #include <string>

// std :: cout

// std :: string

int main ()

{

std :: string str ("There are two needles in this haystack with needles.");

std :: string str2 ("Needle");

// different member versions of find in the same order as above: std :: size\_t found = str.find (str2);

if (Found! = Std :: string :: npos)

std :: cout << "First 'needle' found at:" << found << '\ N';

found = str.find ("Needles are small", Found + 1,6); if (Found! = Std :: string :: npos)

std :: cout << "Second 'needle' found at:" << found << '\ N';

found = str.find ("Haystack");

if (Found! = Std :: string :: npos)

std :: cout << " 'Haystack' also found at:" << found << '\ N';

found = str.find ('.');

if (Found! = Std :: string :: npos)

std :: cout << "Period found at:" << found << '\ N';

// let's replace the first needle:

str.replace (str.find (str2), str2.length (),"Preposition"); std :: cout << str << '\ N';

return 0;

}

Notice how parameter pos is used to search for a second instance of the same search string. Output:

first 'needle' found at: 14 second 'needle' found at: 44 'Haystack' also found at: 30 Period found at: 51

There are two prepositions in this haystack with needles.

// string :: rfind #include <iostream> #include <string> #include <cstddef>

int main ()

{

std :: string str ("The sixth sick sheik's sixth sheep's sick."); std :: string key ("Sixth");

std :: size\_t found = str.rfind (key); if (Found! = Std :: string :: npos)

str.replace (found, key.length (),"Seventh"); std :: cout << str << '\ N';

return 0;

}

The sixth sick sheik's seventh sheep's sick.

// string :: find\_first\_of

#include <iostream> // std :: cout #include <string> // std :: string #include <cstddef> // std :: size\_t

int main ()

{

std :: string str ("Please, replace the vowels in this sentence by asterisks.");

std :: size\_t found = str.find\_first\_of ("Aeiou"); while (Found! = Std :: string :: npos)

{

str [found] ='\*';

found = str.find\_first\_of ("Aeiou", Found + 1);

}

std :: cout << str << '\ N'; return 0;

}

Pl \*\* s \*, r \* pl \* c \* th \* v \* w \* ls \* n th \* ss \* nt \* nc \* by \* st \* r \* sks.

// string :: find\_last\_of

#include <iostream> // std :: cout #include <string> // std :: string

#include <cstddef> // std :: size\_t

void SplitFilename (const std :: string & str)

{

std :: cout << "Splitting:" << str << '\ N';

std :: size\_t found = str.find\_last\_of ("/ \\");

std :: cout << "Path:" << str.substr (0, found) << '\ N'; std :: cout << "File:" << str.substr (found + 1) << '\ N';

}

int main ()

{

std :: string str1 ("/ Usr / bin / man");

std :: string str2 ("C: \\ windows \\ winhelp.exe");

SplitFilename (str1); SplitFilename (str2);

return 0;

}

Splitting: / usr / bin / man path: / usr / bin

file: man

Splitting: c: \ windows \ winhelp.exe

path: c: \ windows file: winhelp.exe

// string :: substr #include <iostream> #include <string>

int main ()

{

std :: string str ="We think in generalities, but we live in details.";

// (quoting Alfred N. Whitehead)

std :: string str2 = str.substr (3,5);

// "think"

std :: size\_t pos = str.find ("Live");

// position of "live" in str

std :: string str3 = str.substr (pos);

// get from "live" to the end

std :: cout << str2 << '' << str3 << '\ N';

return 0;

}

Output:

think live in details.