1.

/\*\*

\* 比较两个字符串的字符序列。

\*

\* @param p1 字符串1的指针。

\* @param p2 字符串2的指针。

\* @return 如果字符串1小于字符串2，则返回负值；如果字符串1大于字符串2，则返回正值；如果两个字符串相等，则返回0。

\*/

int strcmp(const char \*p1, const char \*p2) {

while (\*p1 != '\0' && \*p2 != '\0') {

if (\*p1 > \*p2) return \*p1 - \*p2;

else if (\*p1 < \*p2) return \*p1 - \*p2;

else {

p1++;

p2++;

}

}

if (\*p1 == '\0' && \*p2 == '\0') return 0;

else if (\*p1 == '\0') return -(\*p2);

else return \*p1;

}

/\*\*

\* 比较两个字符串的前n个字符。

\*

\* @param p1 字符串1的指针。

\* @param p2 字符串2的指针。

\* @param n 要比较的字符数。

\* @return 如果字符串1的前n个字符小于字符串2的前n个字符，则返回负值；如果大于，则返回正值；如果相等，则返回0。

\*/

int strncmp(const char \*p1, const char \*p2, int n) {

int i = 0;

while (i < n && \*p1 != '\0' && \*p2 != '\0') {

if (\*p1 > \*p2) return \*p1 - \*p2;

else if (\*p1 < \*p2) return \*p1 - \*p2;

else {

p1++;

p2++;

}

i++;

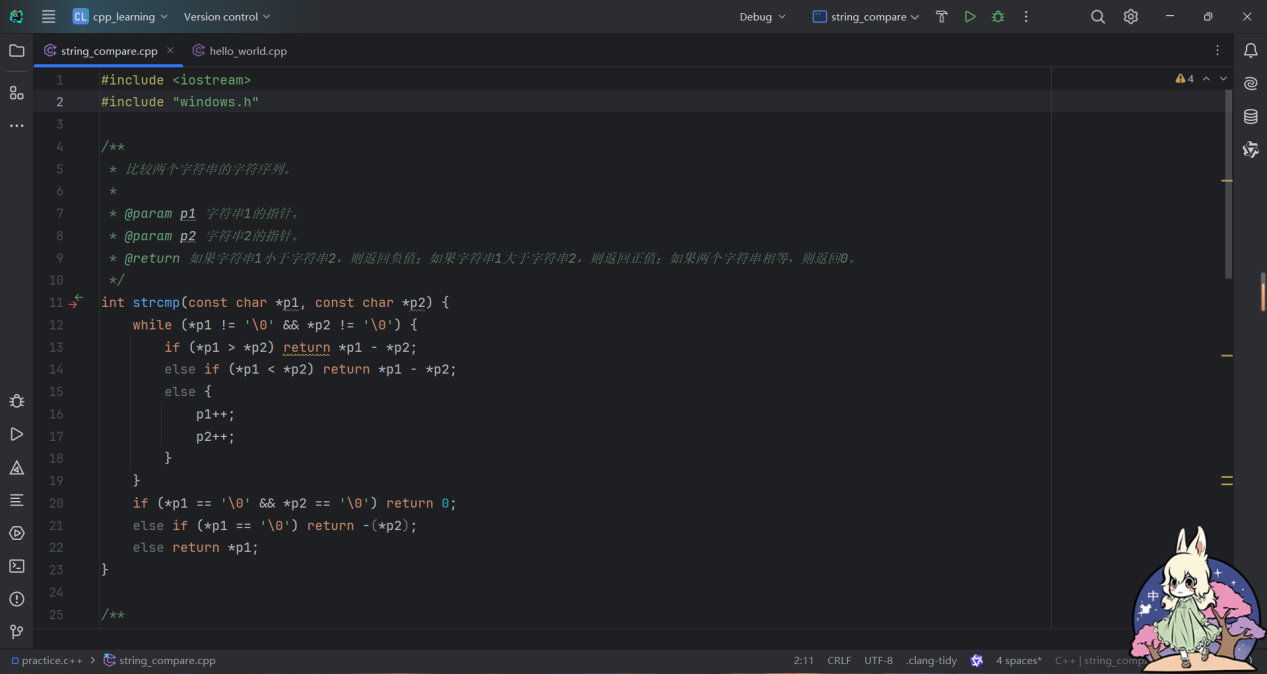
}

if (i == n) return 0;

else if (\*p1 == '\0') return -(\*p2);

else return \*p1;

}



实现结果

// 使用strcmp比较两个字符串的完整内容

int result = strcmp(str1, str2);

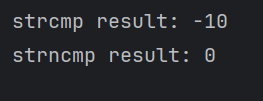
std::cout << "strcmp result: " << result << std::endl;

// 使用strncmp比较两个字符串的前3个字符

int n = 3;

int result\_n = strncmp(str1, str2, n);

std::cout << "strncmp result: " << result\_n << std::endl;



2.

/\*\*

\* 自定义实现字符串连接函数strcat

\* @param dest 目标字符串，将源字符串src追加到该字符串末尾

\* @param src 源字符串，内容将被追加到目标字符串dest末尾

\* @return 返回修改后的目标字符串指针

\*/

// 自己实现的strcat函数

char\* strcat(char \*dest, const char \*src) {

char \*ptr = dest + strlen(dest); // 指针指向dest字符串的末尾

while (\*src != '\0') {

\*ptr++ = \*src++;

}

\*ptr = '\0';

return dest;

}

/\*\*

\* 自定义实现限定长度的字符串连接函数strncat

\* @param dest 目标字符串，将源字符串src的前n个字符追加到该字符串末尾

\* @param src 源字符串，内容将被追加到目标字符串dest末尾

\* @param n 指定追加的字符数上限

\* @return 返回修改后的目标字符串指针

\*/

// 自己实现的strncat函数

char\* strncat(char \*dest, const char \*src, size\_t n) {

char \*ptr = dest + strlen(dest); // 指针指向dest字符串的末尾

int count = 0;

while (\*src != '\0' && count < n) {

\*ptr++ = \*src++;

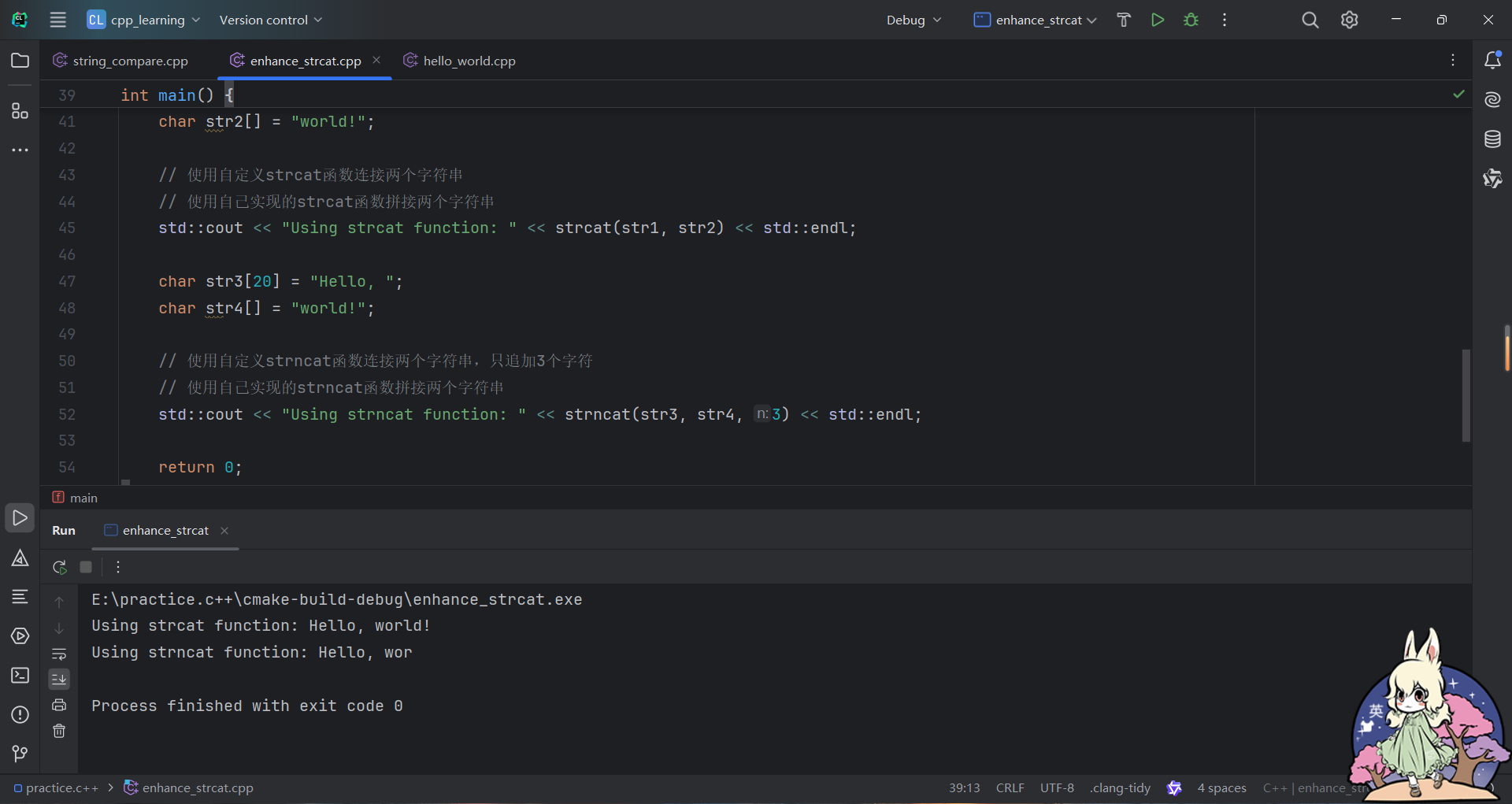
count++;

}

\*ptr = '\0';

return dest;

}



运行结果

int main() {

char str1[20] = "Hello, ";

char str2[] = "world!";

// 使用自定义strcat函数连接两个字符串

// 使用自己实现的strcat函数拼接两个字符串

std::cout << "Using strcat function: " << strcat(str1, str2) << std::endl;

char str3[20] = "Hello, ";

char str4[] = "world!";

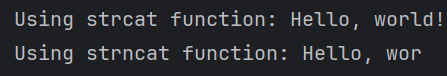
// 使用自定义strncat函数连接两个字符串，只追加3个字符

// 使用自己实现的strncat函数拼接两个字符串

std::cout << "Using strncat function: " << strncat(str3, str4, 3) << std::endl;

return 0;

}



3.

char\* my\_strcpy(char \*dest, const char \*src) {

char \*ptr = dest;

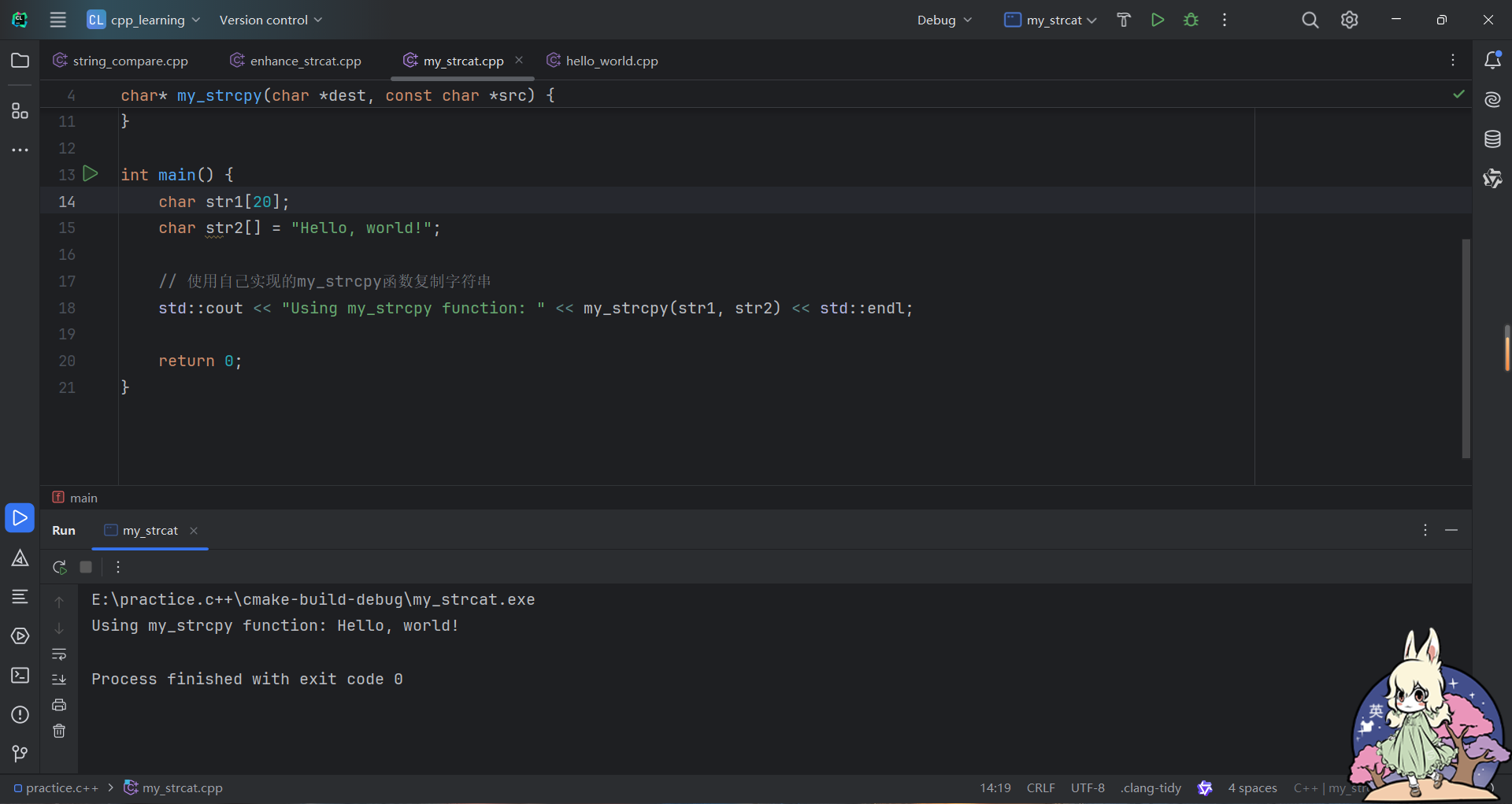
while (\*src != '\0') {

\*ptr++ = \*src++;

}

\*ptr = '\0';

return dest;



运行结果

int main() {

char str1[20];

char str2[] = "Hello, world!";

// 使用自己实现的my\_strcpy函数复制字符串

std::cout << "Using my\_strcpy function: " << my\_strcpy(str1, str2) << std::endl;

return 0;

}



4.

/\*\*

\* 自定义字符串复制函数

\* @param dest 目标字符串数组，用于存放源字符串的内容

\* @param src 源字符串数组，内容将被复制到目标字符串

\* @return 返回指向目标字符串末尾的指针

\*

\* 该函数的功能是将源字符串src的内容复制到目标字符串dest中，

\* 并在复制完成后在目标字符串末尾添加字符串结束符'\0'。

\* 函数返回一个指向目标字符串末尾的指针，可用于后续操作。

\*/

// 自己实现的strcpy函数，返回指向目标字符串末尾的指针

char\* my\_strcpy\_end(char \*dest, const char \*src) {

char \*ptr = dest;

while (\*src != '\0') {

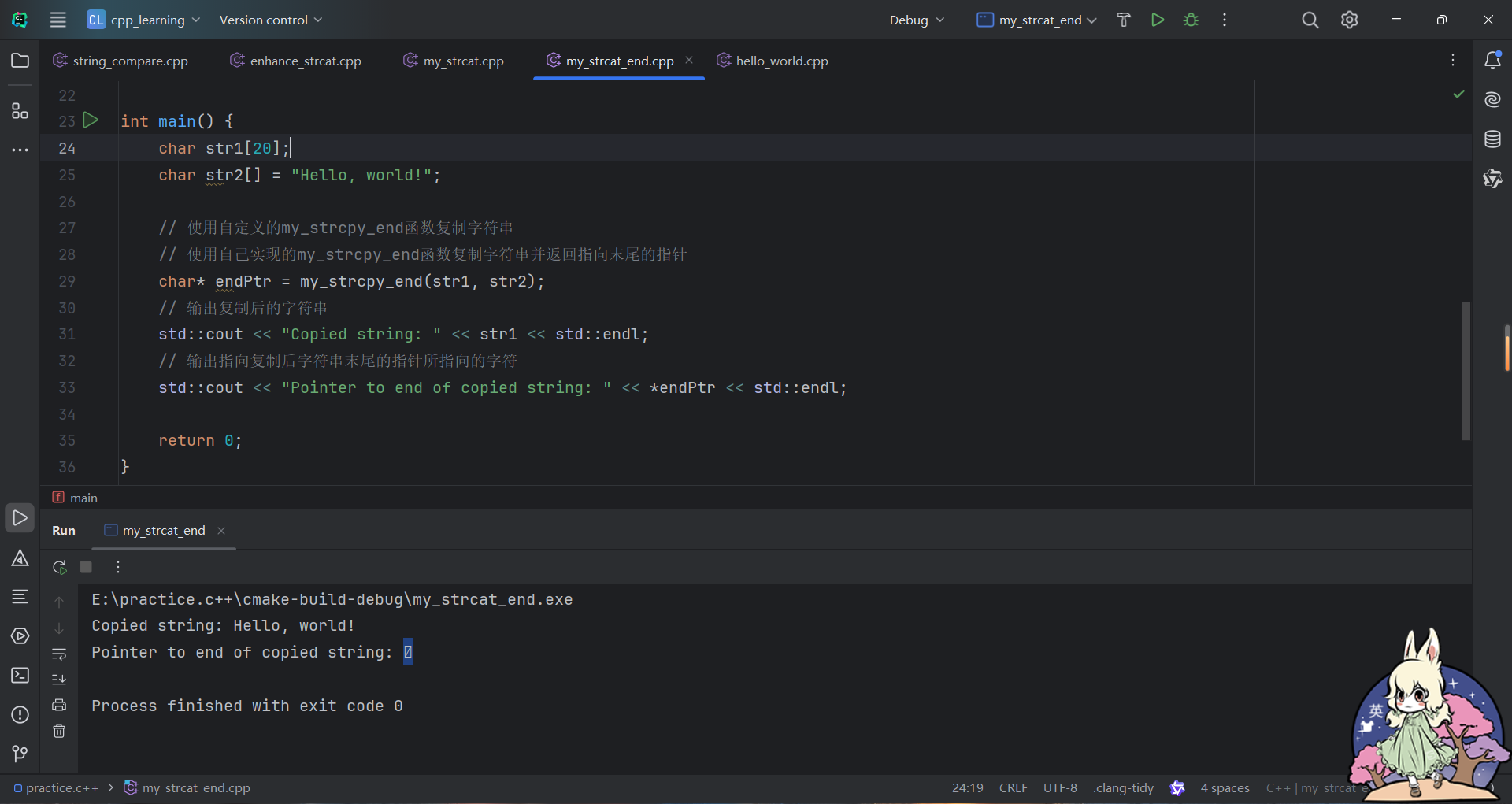
\*ptr++ = \*src++;

}

\*ptr = '\0';

return ptr; // 返回指向目标字符串末尾的指针

}



运行结果

int main() {

char str1[20];

char str2[] = "Hello, world!";

// 使用自定义的my\_strcpy\_end函数复制字符串

// 使用自己实现的my\_strcpy\_end函数复制字符串并返回指向末尾的指针

char\* endPtr = my\_strcpy\_end(str1, str2);

// 输出复制后的字符串

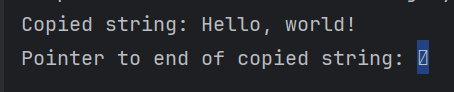
std::cout << "Copied string: " << str1 << std::endl;

// 输出指向复制后字符串末尾的指针所指向的字符

std::cout << "Pointer to end of copied string: " << \*endPtr << std::endl;

return 0;

}



5.

/\*\*

\* 在字符串s1中插入字符串s2。

\* @param s1 原始字符串。

\* @param s2 要插入的字符串。

\* @param n 插入位置的索引，从0开始计数。

\* @return 返回插入后的新的字符串。如果插入位置不合法，返回nullptr。

\*/

char\* insert(char\* s1, const char\* s2, int n) {

// 获取s1的长度

int len1 = strlen(s1);

// 获取s2的长度

int len2 = strlen(s2);

// 检查插入位置是否合法

// 如果n大于s1的长度，返回nullptr

if (n > len1 || n < 0) {

return nullptr;

}

// 动态分配空间存储插入后的字符串

// 创建一个新的字符数组来保存结果

char\* result = new char[len1 + len2 + 1]; // +1 for null-terminator

// 复制s1中前n个字符到result

// 复制s1的前n个字符到result

strncpy(result, s1, n);

// 在result中插入s2

// 将s2插入到result中

strcpy(result + n, s2);

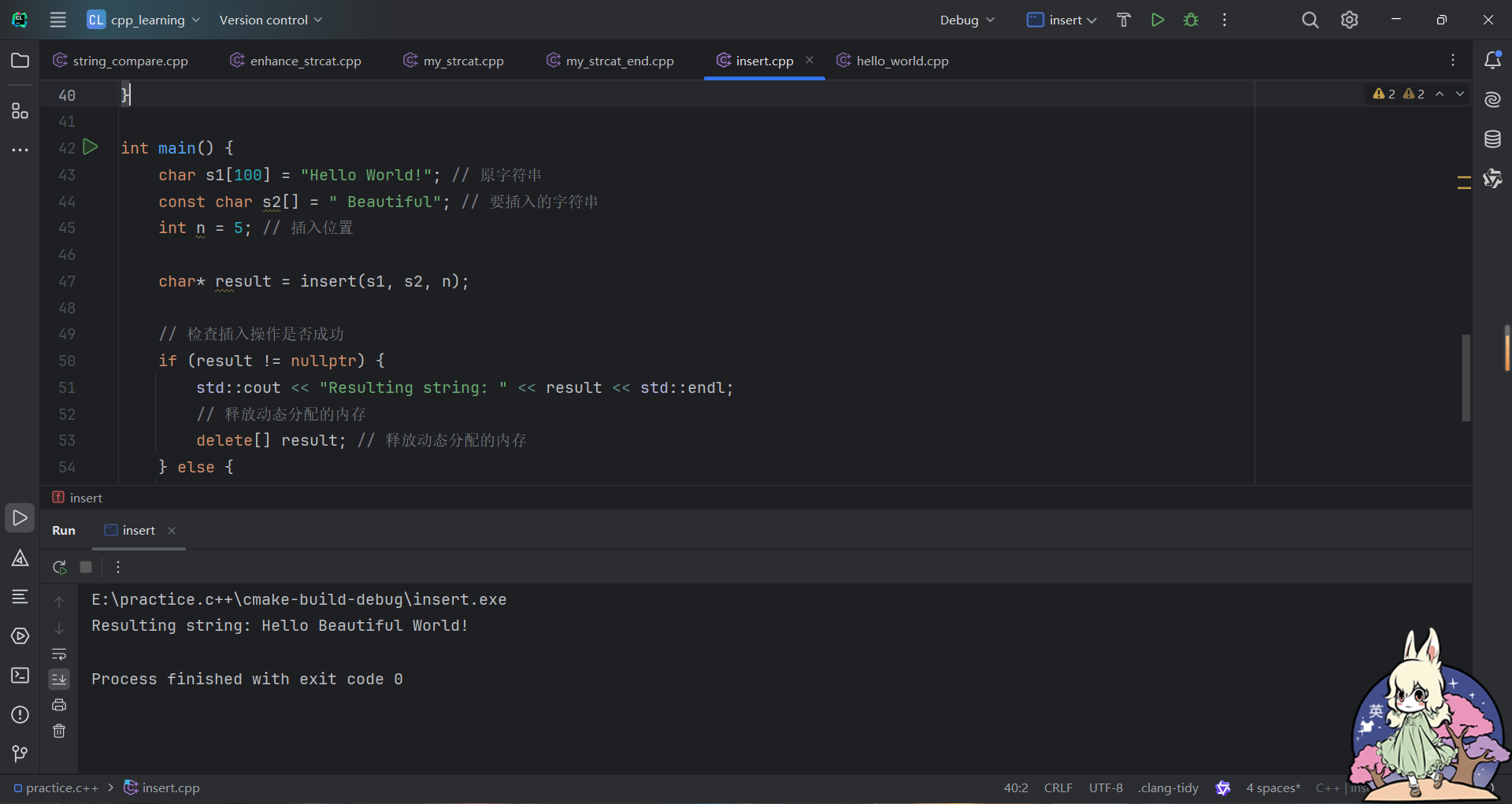
// 复制s1中n之后的字符到result的适当位置

// 继续复制s1剩余的部分到result

strcpy(result + n + len2, s1 + n);

return result; // 返回结果字符串

}



运行结果

int main() {

char s1[100] = "Hello World!"; // 原字符串

const char s2[] = " Beautiful"; // 要插入的字符串

int n = 5; // 插入位置

char\* result = insert(s1, s2, n);

// 检查插入操作是否成功

if (result != nullptr) {

std::cout << "Resulting string: " << result << std::endl;

// 释放动态分配的内存

delete[] result; // 释放动态分配的内存

} else {

std::cout << "Insertion position is out of bounds." << std::endl;

}

return 0;

}



6.

/\*\*

\* 将字符串转换为整数。

\* @param s 输入的字符串，包含待转换的整数。

\* @return 转换后的整数值。

\*/

int atoi(char\* s) {

// 指向当前字符的指针

char\* ptr = s;

// 跳过字符串前的空格

// 跳过前导空格

while (\*ptr && std::isspace(\*ptr)) {

ptr++;

}

// 判断并处理符号位

// 处理符号

bool negative = false;

if (\*ptr == '-') {

negative = true;

ptr++;

} else if (\*ptr == '+') {

ptr++;

}

// 计算数字值

// 处理数字转换

int result = 0;

while (\*ptr && std::isdigit(\*ptr)) {

result = result \* 10 + (\*ptr - '0'); // 计算当前结果

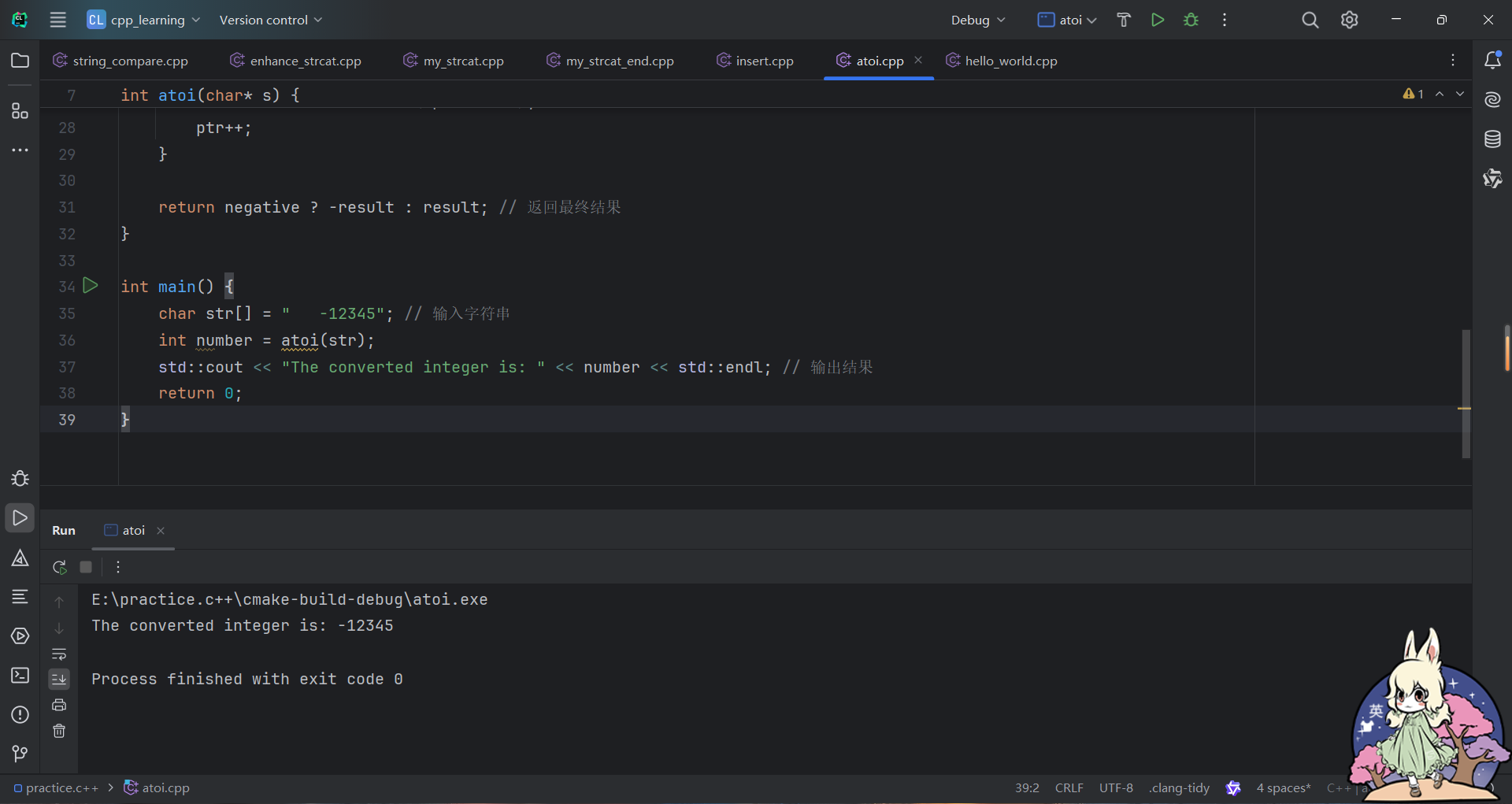
ptr++;

}

// 根据是否有负号返回结果

return negative ? -result : result; // 返回最终结果

}



运行结果

int main() {

char str[] = " -12345"; // 输入字符串

int number = atoi(str);

std::cout << "The converted integer is: " << number << std::endl; // 输出结果

return 0;

}



7.

/\*\*

\* 反转数组元素的顺序。

\* @param a 整型数组，待反转的数组。

\* @param n 整数，数组的长度。

\*/

void reverse(int a[], int n) {

// 使用双指针法进行逆序排列

int start = 0; // 开始指针

int end = n - 1; // 结束指针

while (start < end) {

// 交换元素

int temp = a[start];

a[start] = a[end];

a[end] = temp;

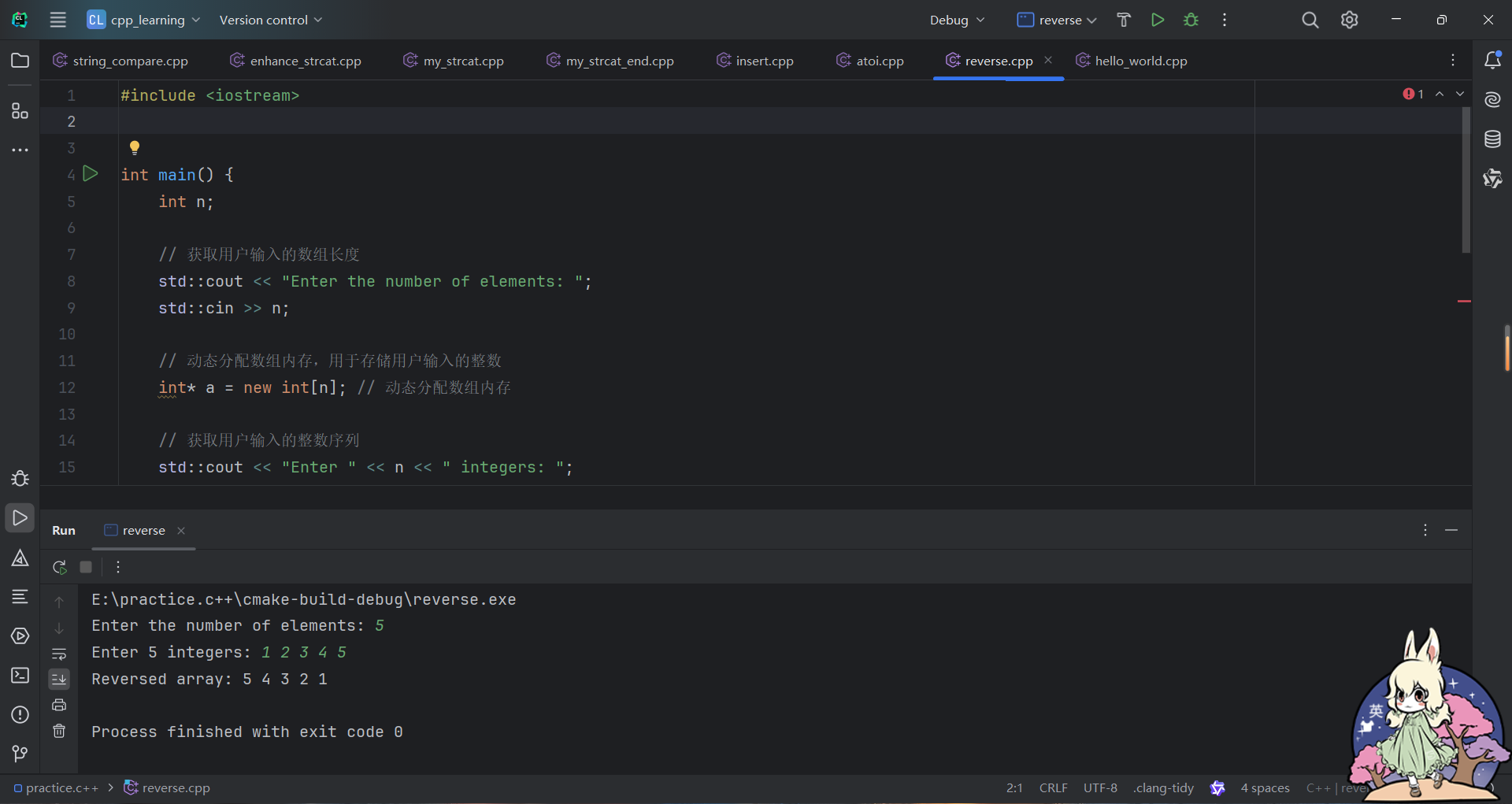
// 移动指针

start++;

end--;

}

}



运行结果

int main() {

int n;

// 获取用户输入的数组长度

std::cout << "Enter the number of elements: ";

std::cin >> n;

// 动态分配数组内存，用于存储用户输入的整数

int\* a = new int[n]; // 动态分配数组内存

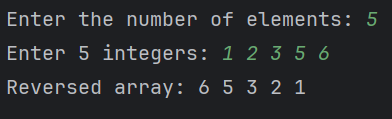
// 获取用户输入的整数序列

std::cout << "Enter " << n << " integers: ";

for (int i = 0; i < n; ++i) {

std::cin >> a[i];

}



8.

void bubbleSort(char\* arr[], int n) {

// 冒泡排序算法实现

for (int i = 0; i < n - 1; ++i) {

for (int j = 0; j < n - i - 1; ++j) {

// 比较字符串并进行排序

if (strcmp(arr[j], arr[j + 1]) > 0) {

// 交换

char\* temp = arr[j];

arr[j] = arr[j + 1];

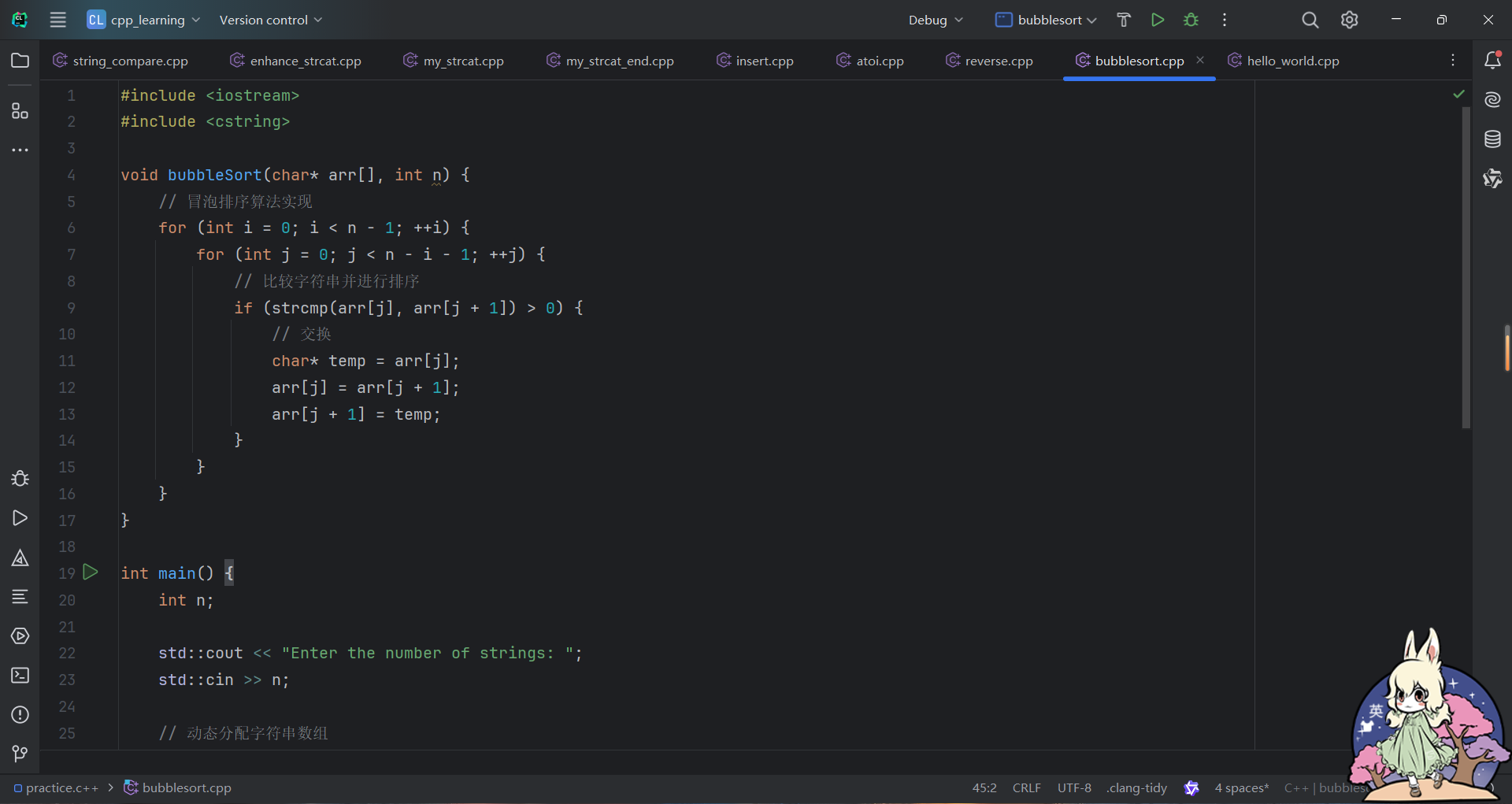
arr[j + 1] = temp;

}

}

}

}



运行结果

int main() {

int n;

std::cout << "Enter the number of strings: ";

std::cin >> n;

// 动态分配字符串数组

char\*\* arr = new char\*[n];

std::cout << "Enter " << n << " strings: " << std::endl;

for (int i = 0; i < n; ++i) {

// 动态分配每个字符串的内存

arr[i] = new char[100]; // 设定每个字符串最大长度为99+1（包含终止符）

std::cin >> arr[i];

}

