در کل، این کامپایلر یک برنامه نوشته شده به زبان X رو به کد معادل در زبان C++ ترجمه میکنه. بیایید قسمت های مختلف این کد رو با هم بررسی کنیم:

**تعریف اولیه**

- این بخش کد کتابخانه های مورد نیاز رو import میکنه.
- همچنین چندتا ثابت عددی(توکن هامون) رو تعریف میکنه که بعدا برای تشخیص نوع داده ها در کد X استفاده میشن (مثلا IN برای کلمه کلیدی `in` و INTEGER برای عدد صحیح).
- در نهایت یک شیء از کلاس Scanner ساخته میشه که برای خواندن ورودی از کاربر استفاده میشه (شبیه به `cin` در C++).

**توابع کمکی**

- `readFile`: این تابع متن یک فایل رو خط به خط میخونه و به صورت یه لیست از رشته ها برمیگردونه.
- `writeFile`: این تابع برعکسِ `readFile` عمل میکنه و یک رشته رو به عنوان محتویات یک فایل جدید ذخیره میکنه.
- `isStringInteger`: این تابع چک میکنه که آیا یک رشته فقط شامل اعداد باشه یا نه (شبیه به `isdigit` در C++).
- `isStringOperand`: این تابع چک میکنه که آیا یک رشته یک متغیر معتبر باشه یا نه (یعنی با حرف شروع بشه و شامل عدد و حرف باشه).
- `lexLine`: این تابع یک خط از کد X رو به عنوان ورودی میگیره و اون رو به لیستی از توکن ها (نشانه ها) تبدیل میکنه. هر توکن نشانه دهنده یک جزء خاص از اون خط مثل کلمه کلیدی، عدد، یا متغیر هست.

**توابع اصلی**

- `parse`: این تابع مهمترین بخش کامپایلر شماست. این تابع لیستی از خطوط کد X رو به عنوان ورودی میگیره و کد معادل C++ رو برمیگردونه.
  - اول یه لیست برای نگهداری اسامی متغیرها تعریف میشه.
  - بعد خط به خط کد X بررسی میشه:
    - اگه خط خالی باشه نادیده گرفته میشه.
    - اگه خط دارای دستور `in` باشه، متغیر مربوطه در کد C++ با `cin` خونده میشه.
    - اگه خط شامل تعریف متغیر باشه، متغیر در کد C++ با `int` تعریف میشه.
    - اگه خط شامل خروجی باشه، خروجی با `cout` در کد C++ چاپ میشه.
    - اگه خط شامل عملیات ریاضی باشه، کد معادل اون با استفاده از اسامی متغیرها در کد C++ ساخته میشه.
  - در نهایت کد C++ ساخته شده همراه با تعریف متغیرها برگردونده میشه.
- `compileFile`: این تابع فایل کد X رو به عنوان ورودی میگیره، کد C++ معادل اون رو تولید میکنه، و اون رو در یک فایل دیگه ذخیره میکنه.

**تابع اصلی**

- این تابع (با اسم `main`) نقطه شروع برنامه هست.
- این تابع بررسی میکنه که آیا تعداد ورودی های داده شده به برنامه درسته هست یا نه (باید شامل مسیر فایل ورودی و خروجی باشه).
- اگه تعداد ورودی ها درست باشه، مسیر فایل ها خونده میشه و تابع `compileFile` برای ترجمه و ذخیره سازی فراخوانی میشه.

**نکات کلی**

- این کامپایلر یک مثال ساده هست و میتونه برای زبان های پیچیده تر با قابلیت های بیشتر گسترش پیدا کنه.
- در این مثال، کامپایلر کد X رو به جای اینکه اجرا کنه، به کد C++ ترجمه میکنه.

**بررسی دقیق تر تابع `parse`**

تابع `parse` در کامپایلر شما نقش کلیدی در ترجمه کد X به C++ داره. بیایید این تابع رو خط به خط بررسی کنیم و عملکرد دقیقش رو توضیح بدیم:

**1. تعریف متغیرها:**

- در ابتدای تابع، یه لیست به نام `symbols` تعریف میشه که برای ذخیره اسامی متغیرها در کد X استفاده میشه.
- یه متغیر به نام `lineCounter` هم برای شمارش خطوط کد X تعریف میشه.

**2. پیمایش خطوط کد X:**

- داخل یه حلقه `for`، خط به خط کد X از لیست `lines` خونده میشه.
- هر خط به عنوان یک رشته (`lineText`) و بعد به عنوان یه آرایه از کلمات (`splittedText`) با استفاده از کاراکتر اسپیس به عنوان جداکننده، پردازش میشه.

**3. تجزیه و تحلیل هر خط:**

- برای هر خط، ابتدا لیست توکن ها (`lexResult`) با استفاده از تابع `lexLine` ساخته میشه. این تابع نوع هر کلمه در خط رو به عنوان عددی (`token`) مشخص میکنه (مثلا `IN` برای کلمه کلیدی `in` و `INTEGER` برای عدد صحیح).
- اگه `lexResult` null باشه، یعنی خط دارای خطای نحوی هست و پیام خطا چاپ میشه و به خط بعدی میره.
- اگه `lexResult` خالی باشه، یعنی خط خالیه و به خط بعدی میره.
- اگه `lexResult` شامل دو عنصر باشه:

- اگه عنصر اول `IN` باشه و عنصر دوم `OPERAND`، یعنی یک دستور `in` وجود داره. در این صورت، متغیر مربوطه با استفاده از `cin` در کد C++ خونده میشه و به لیست `symbols` اضافه میشه.

- اگه `lexResult` شامل بیشتر از دو عنصر باشه:
  - اول بررسی میشه که آیا خط شامل تعریف متغیر هست یا نه:
    - اگه عنصر اول `OPERAND` باشه، یعنی یک متغیر جدید تعریف میشه. در این صورت، اسم متغیر از `splittedText[0]` گرفته میشه و به لیست `symbols` اضافه میشه.
    - کد C++ برای تعریف متغیر با استفاده از `symbol` ساخته میشه.
  - یا اگه عنصر اول `OUT` باشه، یعنی دستور `out` وجود داره. در این صورت، کد C++ برای چاپ خروجی با استفاده از `cout` ساخته میشه.
  - در غیر این صورت، خطا هست و پیام خطا چاپ میشه.
- در ادامه، کد C++ برای محاسبه مقدار متغیر یا انجام عملیات ریاضی ساخته میشه:
  - یه حلقه `for` دیگه داخل حلقه اصلی اجرا میشه تا همه توکن ها در `lexResult` بررسی بشن.
  - متغیر `isFirstItem` برای مشخص کردن اولین توکن در خط استفاده میشه.
  - متغیر `skipFirstEqualSign` برای نادیده گرفتن اولین علامت مساوی در خط (در صورت تعریف متغیر) استفاده میشه.
  - متغیر `mustBeOperator` برای مشخص کردن اینکه توکن بعدی باید عملگر باشه یا نه، استفاده میشه.
  - متغیر `textIndex` برای نگه داشتن موقعیت فعلی در آرایه `splittedText` استفاده میشه.
  - داخل این حلقه:
    - اگه توکن فعلی اولین توکن در خط باشه، به توکن بعدی میره.
    - اگه توکن فعلی علامت مساوی باشه و `skipFirstEqualSign` درست باشه، `skipFirstEqualSign` به False تغییر داده میشه و به توکن بعدی میره.
    - اگه `mustBeOperator` درست باشه و توکن فعلی عملگر نباشه، خطا هست و به توکن بعدی میره.
    - در غیر این صورت:
      - اگه توکن فعلی `OPERAND` باشه، اسم متغیر از `splittedText[textIndex]` گرفته میشه و به کد C++ اضافه میشه.
      - اگه توکن فعلی `INTEGER` باشه، مقدار عدد به کد C++ اضافه میشه.
      - `mustBeOperator` به True تغییر داده میشه.
    - بعد از هر بار اضافه کردن یه جزء به کد C++، `textIndex` یک واحد افزایش پیدا میکنه.
  - در نهایت، کد C++ با نقطه ویرگول تموم میشه.

- و `lineCounter` هم یک واحد زیاد میشه.

**4. ساخت کد C++ برای تعریف متغیرها:**

- بعد از بررسی همه خطوط کد X، لیست `symbols` شامل اسامی تمام متغیرهای استفاده شده در کد X میشه.
- یه رشته جدید به نام `compiledSymbols` ساخته میشه که کد C++ برای تعریف هر متغیر رو نگه میداره.
- برای هر اسم متغیر در `symbols`، یه خط کد C++ با فرمت `int variableName;` به `compiledSymbols` اضافه میشه.

**5. ساخت کد C++ نهایی:**

- یه رشته جدید به نام `compiledResult` ساخته میشه که کد C++ کامل ترجمه شده رو نگه میداره.
- این رشته شامل موارد زیر هست:
  - کتابخانه های مورد نیاز:
  - `# include <iostream>` و `using namespace std;`
  - تابع `main`:
    - شروع تابع با `int main()`.
    - کد C++ برای تعریف متغیرها (`compiledSymbols`).
    - کد C++ برای انجام عملیات و چاپ خروجی (`compiledCode`).
    - دستور `return 0;` برای پایان تابع.
- در نهایت، `compiledResult` برگردونده میشه و به عنوان کد C++ نهایی ذخیره میشه.

**نکات مهم:**

- تابع `parse` از توابع کمکی دیگه مثل `lexLine` برای تجزیه و تحلیل دقیق کد X و ساخت کد C++ معادل استفاده میکنه.
- این تابع به طور دقیق نوع هر جزء در کد X رو بررسی میکنه و کد C++ صحیح و معادل رو تولید میکنه.
- در حال حاضر، این تابع فقط از چند دستور ساده X مثل `in`, `out`, و عملیات ریاضی پایه پشتیبانی میکنه. برای پشتیبانی از دستورات و قابلیت های بیشتر در زبان X، باید تابع `parse` و توابع کمکی مرتبط با اون گسترش داده بشن.

**مثال:**

فرض کنید کد X زیر رو به عنوان ورودی به تابع `parse` میدیم:

```
in a
b = 5

result = a + b

out result
out result + 4
```

در این صورت، تابع `parse` کد C++ معادل زیر رو برمیگردونه:

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

int main()
{
    int a;
    int b;
    int result;

    cin >> a;
    b = 5;
    result = a + b;
    cout << result;
    cout << result + 4;
    return 0;
}
```

همانطور که میبینید، تابع `parse` کد X رو به طور دقیق به کد C++ معادل ترجمه کرده و تمام متغیرها، عملیات، و دستورات رو به درستی در کد C++ پیاده سازی کرده.

**توضیح دقیق تر قسمت 3 (تجزیه و تحلیل هر بخش) تابع `parse`**

قسمت 3 تابع `parse` در کامپایلر شما وظیفه اصلی ترجمه کد X به کد C++ رو بر عهده داره. در این قسمت، کد C++ برای محاسبه مقدار متغیرها یا انجام عملیات ریاضی بر اساس خط فعلی کد X ساخته میشه. بیایید این قسمت رو خط به خط بررسی کنیم و عملکرد دقیقش رو توضیح بدیم:

**1. حلقه `for` دوم:**

- داخل حلقه `for` اصلی، یه حلقه `for` دیگه اجرا میشه تا همه توکن ها در لیست `lexResult` بررسی بشن.
- این حلقه برای هر توکن در خط فعلی کد X اجرا میشه.

**2. بررسی توکن ها:**

- `isFirstItem`:
  - این متغیر برای مشخص کردن اولین توکن در خط استفاده میشه.
  - اگه توکن فعلی اولین توکن باشه، به توکن بعدی میره.
  - این کار برای نادیده گرفتن اولین توکن در خط (که معمولا اسم متغیر یا کلمه کلیدی هست) ضروری هست.
- `skipFirstEqualSign`:
  - این متغیر برای نادیده گرفتن اولین علامت مساوی در خط (در صورت تعریف متغیر) استفاده میشه.

- در خطوطی که متغیر تعریف میشه، اولین علامت مساوی فقط برای جدا کردن اسم متغیر از مقدار اولیه استفاده میشه و در کد C++ نیازی به اون نیست.
- اگه توکن فعلی علامت مساوی باشه و `skipFirstEqualSign` درست باشه، `skipFirstEqualSign` به False تغییر داده میشه و به توکن بعدی میره.

- `mustBeOperator`:
  - این متغیر برای مشخص کردن اینکه توکن بعدی باید عملگر باشه یا نه، استفاده میشه.
  - در خطوطی که شامل عملیات ریاضی هستن، توکن ها به صورت متناوب شامل متغیر، عملگر، متغیر، عملگر و ... هستن.
  - اگه توکن فعلی عملگر نباشه و `mustBeOperator` مقدار True باشه، خطا هست.
- `textIndex`:
  - این متغیر برای نگه داشتن موقعیت فعلی در آرایه `splittedText` استفاده میشه.
  - این متغیر برای دسترسی به مقدار هر توکن در آرایه `splittedText` استفاده میشه.

**3. ساخت کد C++:**

- داخل این حلقه:
  - اگه توکن فعلی `OPERAND` باشه، اسم متغیر از `splittedText[textIndex]` گرفته میشه و به کد C++ اضافه میشه.
    - این کار برای جایگزینی اسم متغیر در کد C++ با مقدار واقعی اون در زمان اجرا انجام میشه.
  - اگه توکن فعلی `INTEGER` باشه، مقدار عدد به کد C++ اضافه میشه.
    - این کار برای اضافه کردن مقادیر عددی ثابت به کد C++ انجام میشه.
  - بعد از هر بار اضافه کردن یه جزء به کد C++، `textIndex` یک واحد افزایش پیدا میکنه.
    - این کار برای حرکت به توکن بعدی در خط فعلی کد X انجام میشه.
  - `mustBeOperator` به True تغییر داده میشه.
    - این کار برای مشخص کردن اینکه توکن بعدی باید عملگر باشه انجام میشه.

**4. تکمیل کد C++:**

- در نهایت، کد C++ برای محاسبه مقدار متغیر یا انجام عملیات ریاضی با نقطه ویرگول ( ;) به پایان میرسه.
  - این کار برای مشخص کردن پایان عبارت در کد C++ انجام میشه.

**مثال:**

فرض کنید خط فعلی کد X به صورت زیر باشه:

```
a = 5 + b
```

در این خط:

- a یک متغیر هست.
- = یک علامت مساوی هست.
- 5 یک عدد صحیح هست.
- + یک عملگر جمع هست.
- b یک متغیر هست.

در این حالت، تابع parse کد C++ زیر رو برای این خط تولید میکنه:

```
int a;
a = 5 + b;
```

**نکات مهم:**

- این قسمت از تابع parse قلب کامپایلر شماست و وظیفه اصلی ترجمه منطق کد X به کد C++ رو بر عهده داره.
- در این قسمت، از یه سری متغیر کمکی و حلقه های for برای بررسی و پردازش توکن ها در خط فعلی کد X استفاده میشه.
- در نهایت، کد C++ معادل برای محاسبه مقدار متغیرها یا انجام عملیات ریاضی ساخته میشه.