پروژه درس مبانی برنامه‌نویسی زبان

C++

دانشکده مهندسی برق

دانشگاه صنعتی شریف


```cpp
int main(){
  cout << "BossFight = Project-IDE" << endl;
  return 0;
}
```

اساتید:

دکتر وثوقی وحدت، دکتر آراسته، دکتر بجانی

پاییز ۱۴۰۳

پیشگفتار

دانشجویان گرامی،

سند حاضر، راهنمای جامع شما برای پیاده‌سازی پروژه پایانی درس مبانی برنامه‌نویسی است. در ابتدا جای تبریک دارد که تا به اینجای کار پیش آمده‌اید - شما به غول مرحله آخر رسیده‌اید! این موفقیت نشان‌دهنده پشتکار و تلاش شما در طول ترم بوده است.
این راهنما که حاصل همفکری جمعی دستیاران آموزشی و بهره‌گیری از نظرات اساتید درس است، با هدف تسهیل مسیر شما در پیاده‌سازی پروژه تهیه شده است. در این سند تلاش کرده‌ایم تمامی جوانب پروژه را به صورت شفاف و گام به گام تشریح کنیم تا بتوانید با درک کامل از خواسته‌های پروژه، بهترین نتیجه را کسب نمایید.
با این حال، از آنجا که هیچ سندی خالی از اشکال نیست، ممکن است در حین کار با سؤالات یا ابهاماتی مواجه شوید. در چنین مواردی، دستیاران آموزشی درس آماده پاسخگویی به سؤالات شما از طریق گروه تلگرامی درس هستند. لطفاً پیش از طرح سؤال، این سند را به دقت مطالعه کرده و در صورت وجود هرگونه ابهام یا پرسش تخصصی، آن را با دستیاران آموزشی در میان بگذارید.

تهیه‌کنندگان سند

این سند به کوشش دستیاران آموزشی پروژه تهیه شده است که اسامی ایشان به ترتیب حروف الفبا در زیر آمده است:

- فاطمه حسن‌پور
- مهدی فلاحی
- امیرمحمد مهرانی کیا
- آمیتیس میرعابدینی
- مریم یاراحمدی
- معین یوسفی‌نیا

با آرزوی موفقیت،
تیم دستیاران آموزشی درس مبانی برنامه‌نویسی

فهرست مطالب

۱ مقدمه

در پروژه پایانی درس برنامه‌نویسی، هدف ما پیاده‌سازی یک محیط توسعه یکپارچه (IDE) ساده است. برای درک بهتر ماهیت این پروژه، می‌توانیم نرم‌افزار CLion را به عنوان یک نمونه حرفه‌ای IDE در نظر بگیریم که اکثر دانشجویان با آن آشنایی دارند. محیط توسعه یکپارچه نرم‌افزاری است که محیطی جامع برای توسعه برنامه‌های کامپیوتری فراهم می‌کند و شامل اجزای اصلی مانند ویرایشگر متن، سیستم مدیریت فایل و رابط کامپایلر می‌باشد. ویرایشگر متن به عنوان هسته اصلی IDE، امکان نوشتن و ویرایش کد را فراهم می‌کند. سیستم مدیریت فایل به کاربر اجازه می‌دهد تا فایل‌های پروژه را به شکلی منظم سازماندهی کند. رابط کامپایلر نیز وظیفه ترجمه کد به زبان ماشین و اجرای برنامه را بر عهده دارد. IDEهای تجاری مانند CLion علاوه بر این قابلیت‌های پایه، امکانات پیشرفته‌تری نیز ارائه می‌دهند، از جمله تکمیل خودکار کد، تشخیص خطا در زمان نگارش، ابزارهای اشکال‌زدایی، مدیریت نسخه‌های کد و ابزارهای تست و پروفایل کد.

در این پروژه، شما با استفاده از کتابخانه SDL و زبان C++ یک IDE ساده برای برنامه‌نویسی به زبان C پیاده‌سازی خواهید کرد. هدف، ساخت یک نمونه آموزشی است که قابلیت‌های پایه‌ای یک IDE را پوشش دهد، نه پیاده‌سازی تمام ویژگی‌های یک IDE حرفه‌ای. برای سهولت در پیاده‌سازی و مدیریت بهتر زمان، پروژه به سه فاز مجزا تقسیم شده است.

در فاز اول، تمرکز بر روی پیاده‌سازی ویرایشگر متن پایه است. در این مرحله، شما باید محیطی برای تایپ و ویرایش متن ایجاد کنید که قابلیت ذخیره و بازیابی فایل‌ها را نیز داشته باشد. فاز دوم به پیاده‌سازی کلیدهای میانبر اختصاص دارد، جایی که عملیات‌های اصلی ویرایش مانند کپی، برش، چسباندن، لغو عملیات و تکرار عملیات را به برنامه اضافه خواهید کرد.

در فاز نهایی، قابلیت‌های پیشرفته‌تر مرتبط با تحلیل و کامپایل کد را پیاده‌سازی خواهید کرد. این شامل تشخیص خطاهای ابتدایی در کد، تبدیل متن به کد قابل کامپایل، برقراری ارتباط با کامپایلر و نمایش نتایج کامپایل و خطاها می‌شود. این پروژه به شما کمک می‌کند درک عمیق‌تری از عملکرد هاIDE به دست آورید و مهارت‌های برنامه‌نویسی خود را در یک پروژه عملی به کار بگیرید.

۲ فاز اول: ساختن Text Editor

۱.۲ رنگ‌آمیزی نحوی کد

در این بخش، هدف این است که نحوه‌ی رنگ‌آمیزی اجزای مختلف زبان C++ با رنگ‌های استفاده شده در محیط توسعه‌ی Visual Studio Code (VSCode) را بررسی کنیم. همانطور که می‌دانید، رنگ‌آمیزی نحوی کد (syntax highlighting) یکی از ویژگی‌های مهم در محیط‌های برنامه‌نویسی است که به افزایش خوانایی و فهم کد کمک می‌کند. در VSCode، هر بخش از کد مانند کلمات کلیدی، توابع، متغیرها، عملگرها و دیگر اجزای زبان، رنگ‌بندی خاص خود را دارند.

جدول رنگ‌بندی حالت روشن (Light Mode)

عنصر	رنگ	Hex	RGB
کلمات کلیدی class, if, while	آبی تیره	#003366	(0, 51, 102)
انواع داده‌ها int, float, double	سبزآبی	#008080	(0, 128, 128)
نام توابع function_name()	نارنجی تیره	#FF8C00	(255, 140, 0)
متغیرها variable_name	قرمز تیره	#B80000	(139, 0, 0)
رشته‌های متنی ”Hello, world!”	سبز تیره	#006400	(0, 100, 0)
کاراکترها ’a’	سبز تیره	#006400	(0, 100, 0)
اعداد 123, 3.14	بنفش	#800080	(128, 0, 128)
توضیحات // comment	خاکستری	#808080	(128, 128, 128)
دستورات پیش‌پردازنده #include	آبی فیروزه‌ای	#0088B8	(0, 139, 139)
عملگرها +, -, *, /	زرشکی	#800000	(128, 0, 0)
پرانتزها و براکت‌ها {}, [], ()	طلایی تیره	#B8860B	(184, 134, 11)

جدول رنگ‌بندی حالت تیره (Dark Mode)

عنصر	رنگ	Hex	RGB
کلمات کلیدی class, if, while	بنفش	#C678DD	(198, 120, 221)
انواع داده‌ها int, float, double	قرمز	#E06C75	(224, 108, 117)
نام توابع function_name()	آبی روشن	#61AFFE	(97, 175, 254)
متغیرها variable_name	زرد	#E5C07B	(229, 192, 123)
رشته‌های متنی ”Hello, world!”	سبز	#98C379	(152, 195, 121)
کاراکترها ’a’	سبز	#98C379	(152, 195, 121)
اعداد 123, 3.14	نارنجی	#D19A66	(209, 154, 102)
توضیحات // comment	خاکستری	#5C6370	(92, 99, 112)
دستورات پیش‌پردازنده #include	آبی فیروزه‌ای	#56B6C2	(86, 182, 194)
عملگرها +, -, *, /	نارنجی تیره	#D55E00	(213, 94, 0)
پرانتزها و براکت‌ها {}, [], ()	خاکستری روشن	#ABB2BF	(171, 178, 191)

۲.۲ کتابخانه‌های مورد نیاز

برای پیاده‌سازی یک Text Editor کارآمد، نیاز به استفاده از کتابخانه‌های مختلف C++ داریم. در این بخش، کتابخانه‌های اصلی مورد نیاز و نحوه استفاده از آن‌ها را بررسی می‌کنیم.

۱.۲.۲ کتابخانه جامع <bits/stdc++.h>

توضیح: این کتابخانه به‌طور غیررسمی و به‌ویژه در کامپایلرهای GCC مانند MinGW برای جمع‌آوری و وارد کردن اکثر کتابخانه‌های استاندارد C++ استفاده می‌شود. اگر این کتابخانه وارد شود، شما می‌توانید از تمامی ویژگی‌های C++ بدون نیاز به وارد کردن جداگانه هر کتابخانه استفاده کنید.

شرایط استفاده:

در صورت وارد کردن این کتابخانه، تمامی توابع و کلاس‌ها به‌طور خودکار در دسترس خواهند بود. برای مثال، می‌توانید از توابع ریاضی موجود در <cmath> بدون هیچ‌گونه محدودیتی استفاده کنید.

کتابخانه‌های شامل شده:

نوع کتابخانه	کتابخانه‌های مربوطه
ورودی/خروجی	<iostream>
ریاضیات	<cmath>
ساختمان داده‌ها	<vector>, <map>, <set>
الگوریتم‌ها	<algorithm>
رشته‌ها	<string>

مثال:

```cpp
#include <bits/stdc++.h>  // Include all standard libraries

int main() {
  double number = 25.0;
  // Using both I/O and math functions
  std::cout << "Square root: " << std::sqrt(number) << std::endl;
  return 0;
}
```

در این مثال، با وجود استفاده از توابع مختلف، نیازی به وارد کردن کتابخانه‌های جداگانه نیست.

۲.۲.۲ کتابخانه <iostream>

توضیح: کتابخانه اصلی برای عملیات ورودی و خروجی در C++. با استفاده از آن می‌توان داده‌ها را از صفحه کلید (ورودی) خواند و نتایج را به صفحه نمایش (خروجی) ارسال کرد.

توابع اصلی:

تابع	عملکرد	مثال
std::cout	خروجی استاندارد	std::cout « ”Hello”;
std::cin	ورودی استاندارد	std::cin » variable;
std::endl	پایان خط	std::cout « std::endl;
std::getline	خواندن یک خط	std::getline(std::cin, str);

شرایط: اگر فقط این کتابخانه وارد شده باشد، نمی‌توانید از توابع cmath استفاده کنید و کد شما به درستی اجرا نخواهد شد.

مثال:

```cpp
#include <iostream>  // Input/Output library only

int main() {
  double number;
  // Reading input
  std::cout << "Enter a number: ";
  std::cin >> number;
  // Writing output
  std::cout << "You entered: " << number << std::endl;
  return 0;
}
```

۳.۲.۲ کتابخانه <cmath>

توضیح: این کتابخانه شامل توابع ریاضی مختلفی است که برای انجام محاسبات ریاضی در C++ استفاده می‌شود. این توابع شامل عملیات‌های پایه مانند سینوس، کسینوس، جذر و توان هستند.

نوع تابع	فرمت	توضیحات
ریشه دوم	std::sqrt(x)	محاسبه ریشه دوم عدد x Example: sqrt(25.0) = 5.0
توان	std::pow(x, y)	محاسبه x به توان y Example: pow(2.0, 3.0) = 8.0
توابع مثلثاتی	std::sin(x) std::cos(x) std::tan(x)	محاسبه سینوس، کسینوس و تانژانت زاویه x (بر حسب رادیان) Example: sin(3.14159/2) ≈ 1.0
قدر مطلق	std::abs(x)	محاسبه قدر مطلق عدد x Example: abs(-5.7) = 5.7
نمایی	std::exp(x)	محاسبه e به توان x Example: exp(1.0) ≈ 2.71828
لگاریتم طبیعی	std::log(x)	محاسبه لگاریتم طبیعی x Example: log(2.71828) ≈ 1.0
لگاریتم مبنای ۱۰	std::log10(x)	محاسبه لگاریتم مبنای ۱۰ عدد x Example: log10(100.0) = 2.0
گرد کردن به پایین	std::floor(x)	گرد کردن به بزرگترین عدد صحیح کوچکتر مساوی x Example: floor(3.7) = 3.0
گرد کردن به بالا	std::ceil(x)	گرد کردن به کوچکترین عدد صحیح بزرگتر مساوی x Example: ceil(3.2) = 4.0

شرایط: اگر تنها این کتابخانه وارد شود، توابع ریاضی در دسترس خواهند بود اما اگر بخواهید از سایر توابع مانند cout استفاده کنید، برنامه به درستی اجرا نخواهد شد.

۳.۲ ذخیره فایل و نمودار درختی

در این بخش، نحوه پیاده‌سازی سیستم مدیریت فایل‌ها را بررسی می‌کنیم. این سیستم شامل قابلیت ذخیره‌سازی پروژه‌ها و نمایش آن‌ها در یک ساختار درختی است.

۱.۳.۲ دکمه ذخیره پروژه

- در سمت راست صفحه، یک دکمه با عنوان «ذخیره» طراحی کنید.
- با کلیک روی این دکمه، از کاربر خواسته می‌شود تا یک نام برای پروژه خود وارد کند.
- پس از وارد کردن نام، پروژه در لیست پروژه‌ها که به صورت یک لیست در یک میره شود درختی است ذخیره می‌شود.

۲.۳.۲ لیست پروژه‌ها

- در سمت چپ صفحه، یک لیست عمودی (به صورت درختی) نمایش داده شود که شامل نام تمامی پروژه‌های ذخیره شده است.
- هر بار که یک پروژه جدید ذخیره می‌شود، نام آن به لیست اضافه شود.
- اگر لیست خالی است، پیامی مانند «هیچ پروژه‌ای ذخیره نشده است» نمایش داده شود.

۳.۳.۲ ویژگی های امتیازی

کاربر باید بتواند با کلیک روی هر پروژه در لیست آن پروژه را باز کند.

هنگام باز کردن پروژه، پروژه فعلی بسته می‌شود و پروژه جدید به عنوان پروژه جاری نمایش داده می‌شود.

۴.۲ اهداف فاز اول

هدف این بخش بطور کلی اجرا و پیاده سازی موارد زیر می باشد:

۱.۴.۲ Highlighting Syntax

۲.۴.۲ mode Light

- رنگ پس‌زمینه: سفید
- رنگ متن: مشکی

۳.۴.۲ mode Dark

- رنگ پس‌زمینه: سیاه
- رنگ متن: سفید

۵.۲ کتابخانه‌ها

- `<bits/stdc++.h>`
- `<iostream>`
- `<cmath>`

۶.۲ ذخیره فایل و نمودار درختی

- دکمه ذخیره: در سمت راست صفحه قرار گرفته و امکان نام‌گذاری پروژه را فراهم می‌کند.
- لیست پروژه‌ها: در سمت چپ صفحه نمایش داده می‌شود و شامل تمامی پروژه‌های ذخیره‌شده است.

۷.۲ منوبار

منوبار شامل گزینه‌های زیر است:

- **File**
  - Project New
  - Project Save
  - Exit
- **Edit**
  - Undo
  - Redo(امتیازی)
- **View**
  - Toggle Dark/Light mode
- **Debug & Compile**
- **Run**

۸.۲ تکمیل خودکار

برنامه بایستی بتواند پرانتز ها ، کروشه ها، و... را تشخیص داده و خودکار کامل کند.
همچنین بصورت امتیازی می توان موضوعات زیر را پیاده سازی کرد:

- کلمات کلیدی (`Keywords`): switch while، else، if،
- توابع (`Functions`): strlen scanf، printf،
- کلمات عمومی (`words General`): float int، main،

۹.۲ پیاده‌سازی ویرایشگر گرافیکی نمونه

همراه پروژه، یک تکست ادیتور خام نیز به شما داده می شود که می توانید از آن به عنوان نقطه شروع کار خود استفاده کنید. در این بخش قصد داریم تا به معرفی نحوه کارکرد کد های این ادیتور بپردازیم. توصیه می شود حتی اگر قصد دارید از این کد استفاده نکنید، حتما این بخش را مطالعه کنید چون دید خوبی برای نحوه برخورد با کتابخانه sdl به شما می دهد.

۱.۹.۲ ساختارهای داده اصلی

برای نگهداری و مدیریت متن، از چند متغیر کلیدی استفاده می‌کنیم:

```
// Store each line of text
std::vector<std::string> lines = {""};

// Track cursor position
int currentLine = 0;    // Current line number
int cursorPos = 1;      // Position in current line
```

برای مدیریت نمایش و اسکرول:

```
// Scrolling and display
int scrollOffset = 0;   // Pixels scrolled from top
const int LINE_HEIGHT = TTF_FontHeight(font);
```

نکات مهم در مورد این ساختار:

- بردار lines به ما امکان مدیریت جداگانه هر خط را می‌دهد
- از currentLine و cursorPos برای کنترل دقیق مکان‌نما استفاده می‌کنیم
- scrollOffset برای نمایش متن‌های طولانی‌تر از صفحه استفاده می‌شود

۲.۹.۲ پردازش ورودی

هنگام حذف متن (Backspace):

```
if (e.key.keysym.sym == SDLK_BACKSPACE) {
  if (cursorPos > 1 && cursorPos <= lines[currentLine].size()
  ) {
    // Remove character before cursor
    lines[currentLine].erase(cursorPos - 1, 1);
    cursorPos--;
  } else if (currentLine > 0) {
```

```
      // Merge with previous line
      cursorPos = lines[currentLine - 1].size();
      lines[currentLine - 1] += lines[currentLine].substr(1);
      lines.erase(lines.begin() + currentLine);
      currentLine--;
    }
  }
```

این کد دو حالت را مدیریت می‌کند:

- حذف یک کاراکتر از وسط خط
- ادغام خط جاری با خط قبلی (وقتی مکان‌نما در ابتدای خط است)

هنگام ایجاد خط جدید (Enter):

```
  if (e.key.keysym.sym == SDLK_RETURN) {
    if (cursorPos <= lines[currentLine].size()) {
      // Split line at cursor
      std::string remainder = lines[currentLine].substr(
    cursorPos);
      lines[currentLine] = lines[currentLine].substr(0,
    cursorPos);
      lines.insert(lines.begin() + currentLine + 1, remainder);
      currentLine++;
      cursorPos = 0;
    }
  }
```

این کد خط جاری را در محل مکان‌نما به دو قسمت تقسیم می‌کند.

۳.۹.۲ نمایش متن

فرایند نمایش متن شامل سه مرحله اصلی است:

۱. **پاک کردن صفحه:**

```
  // Set background color to white
  SDL_SetRenderDrawColor(renderer, 255, 255, 255, 255);
  SDL_RenderClear(renderer);
```

۲. **رندر کردن متن:**

```
  int y = -scrollOffset;
  for (size_t i = 0; i < lines.size(); ++i) {
    if (y + LINE_HEIGHT > 0 && y < SCREEN_HEIGHT) {
```

```
        // Create text texture
        SDL_Surface* textSurface = TTF_RenderText_Blended(
        font, lines[i].c_str(), textColor);
        SDL_Texture* textTexture =
        SDL_CreateTextureFromSurface(renderer, textSurface);

        // Render text line
        SDL_Rect renderQuad = {10, y, textWidth, textHeight};
        SDL_RenderCopy(renderer, textTexture, nullptr, &
      renderQuad);

        // Clean up
        SDL_FreeSurface(textSurface);
        SDL_DestroyTexture(textTexture);
      }
      y += LINE_HEIGHT;
    }
```

۳. **نمایش مکان‌نما:**

```
    if (i == currentLine) {
      int cursorX = 0;
      TTF_SizeText(font, lines[i].substr(0, cursorPos).c_str
      (),
      &cursorX, nullptr);
      cursorX += 10;  // Add left margin
      SDL_RenderDrawLine(renderer, cursorX, y,
      cursorX, y + LINE_HEIGHT);
    }
```

نکات مهم در مورد رندرینگ:

- تنها خطوط قابل مشاهده را رندر می‌کنیم تا عملکرد بهتری داشته باشیم
- موقعیت مکان‌نما بر اساس عرض متن قبل از آن محاسبه می‌شود
- بعد از هر رندر، منابع SDL را آزاد می‌کنیم تا از نشت حافظه جلوگیری شود

۴.۹.۲ مدیریت اسکرول

برای اطمینان از قابل مشاهده بودن متن، از تابع زیر استفاده می‌کنیم:

```
    void ensureLastLineVisible(int currentLine, int &scrollOffset
      ,
```

```
  int SCREEN_HEIGHT, int LINE_HEIGHT, int totalLines) {

    // Calculate cursor position relative to view
    int cursorY = currentLine * LINE_HEIGHT - scrollOffset;

    // Scroll if cursor is out of view
    if (cursorY < 0) {
      scrollOffset = currentLine * LINE_HEIGHT;
    } else if (cursorY + LINE_HEIGHT > SCREEN_HEIGHT) {
      scrollOffset = (currentLine + 1) * LINE_HEIGHT
      - SCREEN_HEIGHT;
    }

    // Handle documents shorter than screen
    int contentHeight = totalLines * LINE_HEIGHT;
    if (contentHeight > SCREEN_HEIGHT) {
      scrollOffset = std::min(scrollOffset,
      contentHeight - SCREEN_HEIGHT);
    } else {
      scrollOffset = 0;
    }
  }
```

این تابع:

- موقعیت مکان‌نما نسبت به پنجره را محاسبه می‌کند
- در صورت خارج شدن مکان‌نما از دید، صفحه را اسکرول می‌کند
- برای متون کوتاه‌تر از صفحه، اسکرول را غیرفعال می‌کند
- از اسکرول بیش از حد جلوگیری می‌کند

۵.۹.۲ نکات پیاده‌سازی

برای استفاده و توسعه این کد، به موارد زیر توجه کنید:

- **مدیریت حافظه:** منابع SDL (مانند Surface و Texture) باید بعد از استفاده آزاد شوند.
- **بهینه‌سازی:** فقط خطوط قابل مشاهده را رندر می‌کنیم تا عملکرد بهتری داشته باشیم.
- **کنترل خطا:** همیشه خروجی توابع SDL را بررسی کنید و در صورت خطا، پیام مناسب نمایش دهید.
- **به‌روزرسانی مکان‌نما:** هر تغییر در متن باید با به‌روزرسانی موقعیت مکان‌نما همراه باشد.

۳ فاز دوم: پیاده‌سازی کلیدهای میانبر

در این بخش به بررسی و پیاده‌سازی کلیدهای میانبر (Shortcuts) در Text Editor می‌پردازیم. کلیدهای میانبر به کاربر امکان می‌دهند تا عملیات‌های پرکاربرد را با سرعت و سهولت بیشتری انجام دهد.

۱.۳ کلیدهای میانبر اصلی

در ادامه، مهم‌ترین کلیدهای میانبر و عملکرد آن‌ها معرفی می‌شوند:

کلیدهای میانبر پایه

کلید میانبر	عملکرد
Ctrl + C (copy)	با فشردن این دکمه‌ها باید بتوان متن انتخاب شده را کپی کرد.
Ctrl + V (paste)	با فشردن این دکمه‌ها باید بتوان متن کپی شده در کلیپ‌بورد را پیست کرد.
Ctrl + X (cut)	با فشردن این دکمه‌ها باید بتوان متن انتخاب شده را کات کرد.
Ctrl + A (select all)	با فشردن این دکمه‌ها باید بتوان تمام متن را سلکت کرد.
Ctrl + Z (undo)	با فشردن این دکمه‌ها باید بتوان عملیات انجام شده را بازگرداند.
Ctrl + G (go to line)	با فشردن این دکمه‌ها باید بتوان با وارد کردن شماره خط به همان خط در برنامه رفت.
Ctrl + S (save/save as)	با فشردن این دکمه‌ها باید بتوان فایل را در همان محل باز کردن برنامه ذخیره کرد. نیازی به تعیین محل ذخیره‌سازی توسط کاربر نمی‌باشد.

۲.۳ ویژگی امتیازی

پیاده‌سازی Save As در Ctrl + S برای انتخاب مسیر ذخیره‌سازی توسط کاربر

۴ فاز سوم: پیاده‌سازی بخش Debugging

در این بخش به پیاده‌سازی قابلیت‌های اشکال‌یابی و تشخیص خطاهای سینتکسی می‌پردازیم. این قابلیت‌ها باید قبل از ارسال کد به کامپایلر، خطاهای احتمالی را شناسایی و به کاربر گزارش دهند.

۱.۴ تشخیص خطاهای سینتکسی

برنامه باید قادر به تشخیص و نمایش خطاهای سینتکسی زیر باشد:

خطاهای سینتکسی پایه

نوع خطا	توضیحات و مثال
عدم وجود semicolon	عدم وجود ; در پایان دستورات
خطای پرانتزگذاری	عدم تطابق در باز و بسته شدن ()، []، {}
خطای املایی کلمات کلیدی	مثال: whiel به جای while
نامگذاری نادرست متغیرها	استفاده از کلمات کلیدی برای نام متغیر مثال: int return;
متغیر تعریف نشده	استفاده از متغیری که قبلاً تعریف نشده است
خطای رشته	عدم استفاده صحیح از ” ” برای رشته‌ها
عملگرهای نامعتبر	استفاده از عملگرهای تعریف نشده مثال: +== یا +++
پارامترهای تابع	عدم تطابق تعداد پارامترها در فراخوانی تابع
کامنت‌های چندخطی	عدم بسته شدن صحیح کامنت‌ها با */

۲.۴ نکات پیاده‌سازی

در پیاده‌سازی بخش debugging باید به نکات زیر توجه شود:

- تمام خطاها باید قبل از ارسال کد به کامپایلر شناسایی و نمایش داده شوند.
- در صورت وجود چندین خطا، تمامی موارد باید به کاربر گزارش شوند.
- پیام‌های خطا باید واضح و راهنماکننده باشند.

۳.۴ (پیاده سازی این روش ها به صورت امتیازی می باشد) قابلیت‌های پیشرفته

علاوه بر تشخیص خطاها، قابلیت‌های پیشرفته زیر نیز به صورت امتیازی قابل پیاده‌سازی هستند:

قابلیت	توضیحات
تشخیص if-else تکراری	تشخیص if-elseهای متوالی و پیشنهاد استفاده از switch-case
جایگزینی خودکار	تبدیل خودکار if-elseهای متوالی به ساختار switch-case
بهینه‌سازی کد تکراری	تشخیص بلاک‌های تکراری و ارائه پیشنهاد برای کوتاه‌سازی کد

۱.۳.۴ مثال بهینه‌سازی کد

کد اولیه:

```
cout << "1" << endl;
cout << "2" << endl;
cout << "3" << endl;

```

کد بهینه شده:

```
for (int i = 1; i <= 3; i++) {
  cout << i << endl;
}

```

۵ ضمیمه مهم: کار با کامپایلر و ساخت IDE نمونه

۱.۵ آشنایی با کامپایلر GCC

در این بخش، با کامپایلر GCC و دستورات پایه‌ای آن آشنا می‌شویم. کامپایلر GCC یکی از پرکاربردترین کامپایلرهای زبان C++ است که به صورت رایگان در دسترس برنامه‌نویسان قرار دارد.

۱.۱.۵ نصب و راه‌اندازی GCC

قبل از شروع کار با GCC، باید از نصب بودن آن روی سیستم خود اطمینان حاصل کنیم. برای بررسی نصب بودن GCC، می‌توانیم در ترمینال دستور زیر را اجرا کنیم:

```
g++ --version

```

اگر GCC نصب باشد، اطلاعاتی مانند نسخه آن نمایش داده می‌شود. در غیر این صورت، باید آن را نصب کنیم:

۱. دانلود و نصب MinGW از سامانه درس افزار

۲. اضافه کردن مسیر نصب به Path سیستم

۳. تأیید نصب با دستور g++ –version

۲.۱.۵ دستورات پایه‌ای GCC

در این بخش با دستورات اصلی GCC که برای ساخت یک IDE ساده به آن‌ها نیاز داریم، آشنا می‌شویم:

دستور	توضیحات
g++ file.cpp	کامپایل فایل و ایجاد خروجی با نام پیش‌فرض a.out (در لینوکس) یا a.exe (در ویندوز)
g++ -o output.exe file.cpp	کامپایل فایل و ایجاد خروجی با نام دلخواه
g++ -Wall file.cpp	کامپایل با نمایش تمام هشدارها
g++ -std=c++17 file.cpp	کامپایل با استاندارد مشخص C++

مثال ۱: کامپایل یک برنامه ساده
ابتدا یک فایل با نام hello.cpp ایجاد می‌کنیم:

```cpp
#include <iostream>

int main() {
  std::cout << "Hello, World!" << std::endl;
  return 0;
}

```

حال می‌توانیم این برنامه را با دستورات مختلف کامپایل کنیم:

```
# Compile with default output name
g++ hello.cpp

# Compile with specific output name
g++ -o hello.exe hello.cpp

# Compile with warnings enabled
g++ -Wall -o hello.exe hello.cpp

```

۳.۱.۵ پرچم‌های مهم GCC

پرچم‌ها (flags) گزینه‌هایی هستند که رفتار کامپایلر را کنترل می‌کنند. در اینجا با مهم‌ترین پرچم‌ها آشنا می‌شویم:

پرچم	توضیحات
-Wall	فعال‌سازی تمام هشدارهای متداول
-Werror	تبدیل تمام هشدارها به خطا
-std=c++XX	تعیین نسخه استاندارد C++ (مثلاً c++11، c++14، c++17)
-O0, -O1, -O2, -O3	سطوح مختلف بهینه‌سازی کد (از بدون بهینه‌سازی تا حداکثر بهینه‌سازی)
-g	اضافه کردن اطلاعات دیباگ به فایل اجرایی
-I	تعیین مسیر برای جستجوی فایل‌های هدر
-L	تعیین مسیر برای جستجوی کتابخانه‌ها
-l	مشخص کردن کتابخانه‌های مورد نیاز

مثال ۲: استفاده از پرچم‌های مختلف
برنامه‌ای با چند هشدار احتمالی:

```
#include <iostream>

int main() {
  int x;  // Uninitialized variable
  if(x == 0) {  // Using uninitialized variable
    std::cout << "x is zero" << std::endl;
  }
  return 0;
}

```

کامپایل با پرچم‌های مختلف:

```
# Compile without warnings
g++ warning_example.cpp -o prog

# Compile with warnings enabled
g++ -Wall warning_example.cpp -o prog

# Compile with warnings as errors
g++ -Wall -Werror warning_example.cpp -o prog

```

۴.۱.۵ مدیریت چندین فایل منبع

در پروژه‌های واقعی، معمولاً کد ما در چندین فایل مختلف قرار دارد. در اینجا نحوه کامپایل پروژه‌های چند فایلی را بررسی می‌کنیم.

مثال ۳: پروژه چند فایلی
ساختار فایل‌ها:

```
project/
├── include/
│   └── calculator.h
├── src/
│   ├── calculator.cpp
│   └── main.cpp

```

محتوای فایل‌ها:
calculator.h:

```
#ifndef CALCULATOR_H
#define CALCULATOR_H

class Calculator {
  public:
  static int add(int a, int b);
  static int subtract(int a, int b);
};

#endif

```

calculator.cpp:

```
#include "../include/calculator.h"

int Calculator::add(int a, int b) {
  return a + b;
}

int Calculator::subtract(int a, int b) {
  return a - b;
}

```

main.cpp:

```
#include <iostream>
#include "../include/calculator.h"

int main() {
  std::cout << "5 + 3 = " << Calculator::add(5, 3) << std::
  endl;
  std::cout << "5 - 3 = " << Calculator::subtract(5, 3) <<
  std::endl;
  return 0;
}

```

کامپایل پروژه:

```
# Compile source files into object files separately
g++ -c src/calculator.cpp -o calculator.o -I include
g++ -c src/main.cpp -o main.o -I include
```

```
4
5   # Link object files to create executable
6   g++ calculator.o main.o -o calculator
7
```

در این مثال:

- از پرچم c- برای کامپایل بدون لینک کردن استفاده می‌کنیم
- پرچم I- مسیر فایل‌های هدر را مشخص می‌کند
- فایل‌های o. فایل‌های شیء هستند که باید با هم لینک شوند
- دستور آخر فایل‌های شیء را لینک کرده و فایل اجرایی نهایی را می‌سازد

۲.۵ ارتباط با سیستم‌عامل در C++

در این بخش، با روش‌های تعامل با سیستم‌عامل ویندوز و مفاهیم پایه‌ای کار با فایل در C++ آشنا می‌شویم.

۱.۲.۵ اجرای دستورات سیستمی با system()

تابع system() در C++ امکان اجرای دستورات cmd را فراهم می‌کند. در ویندوز، می‌توانیم از این تابع برای اجرای دستورات کامپایلر و مدیریت فایل‌ها استفاده کنیم.

```
1   #include <iostream>
2   #include <cstdlib>
3   #include <string>
4
5   bool execute_command(const std::string& cmd) {
6     return (system(cmd.c_str()) == 0);
7   }
8
9   int main() {
10    // Create a directory
11    execute_command("mkdir test_folder");
12
13    // Run program in new window
14    execute_command("start cmd /c program.exe");
15
16    return 0;
17  }
```

در کد بالا چند نکته مهم وجود دارد:

- تابع c_str(): این تابع رشته‌های C++ را به رشته‌های C تبدیل می‌کند. تابع system() نیاز به رشته‌های C دارد.
- مقدار بازگشتی system(): این تابع در صورت موفقیت مقدار ۰ و در صورت شکست مقدار غیر صفر برمی‌گرداند.
- cmd /c: این پارامتر در ویندوز به معنی اجرای دستور و بسته شدن پنجره پس از اتمام آن است.

۲.۲.۵ خواندن خروجی دستورات

برای خواندن خروجی دستورات سیستمی، از تابع _popen در ویندوز استفاده می‌کنیم. این تابع یک جریان را برای خواندن خروجی دستور باز می‌کند.

```cpp
#include <iostream>
#include <string>

std::string get_command_output(const std::string& cmd) {
  std::string result;
  char buffer[128];

  // Open pipe to command
  FILE* pipe = _popen(cmd.c_str(), "r");
  if (!pipe) {
    return "Error executing command.";
  }

  // Read output line by line
  while (fgets(buffer, sizeof(buffer), pipe)) {
    result += buffer;
  }

  // Close pipe
  _pclose(pipe);
  return result;
}

int main() {
  // Example: getting compiler version
  std::string output = get_command_output("g++ --version");
  std::cout << output;
  return 0;
}
```

توضیح مفاهیم جدید در کد بالا:

- ()popen_: این تابع یک دستور را اجرا کرده و یک کانال ارتباطی (pipe) برای خواندن خروجی آن ایجاد می‌کند. پارامتر "r" به معنی خواندن (read) از این کانال است.
- *FILE: این نوع داده یک اشاره‌گر به ساختار فایل در C++ است که برای کار با فایل‌ها و جریان‌های ورودی/خروجی استفاده می‌شود.
- ()fgets: این تابع یک خط از ورودی را می‌خواند و در یک بافر ذخیره می‌کند. پارامترهای آن به ترتیب:
  - بافر برای ذخیره داده
  - اندازه بافر
  - اشاره‌گر به فایل یا جریان ورودی
- ()pclose_: این تابع کانال ارتباطی را می‌بندد و منابع سیستمی را آزاد می‌کند.

۳.۲.۵ کار با فایل‌ها در C++

برای کار با فایل‌ها در C++، از کلاس‌های ofstream (نوشتن) و ifstream (خواندن) استفاده می‌کنیم.

توابع پایه‌ای کار با فایل

تابع	توضیحات
open()	باز کردن فایل
is_open()	بررسی باز بودن فایل
close()	بستن فایل
getline()	خواندن یک خط از فایل

مثال ۱: نوشتن در فایل

```
std::ofstream file("output.txt", std::ios::app);
if (file.is_open()) {
  file << "Writing to file" << std::endl;
  file.close();
}
```

در کد بالا:

- ofstream: کلاسی برای نوشتن در فایل است (output file stream).
- std::ios::app: یک پرچم است که نشان می‌دهد می‌خواهیم به انتهای فایل اضافه کنیم (append). بدون این پرچم، محتوای قبلی فایل پاک می‌شود.
- عملگر »: همانند cout برای نوشتن در فایل استفاده می‌شود.

مثال ۲: خواندن از فایل

```
  std::ifstream file("input.txt");
  std::string line;
  if (file.is_open()) {
    while (std::getline(file, line)) {
      std::cout << line << std::endl;
    }
    file.close();
  }
```

در کد بالا:

- ifstream: کلاسی برای خواندن از فایل است (input file stream).
- getline(): تابعی برای خواندن یک خط کامل از فایل است. پارامترهای آن:
  - جریان ورودی (file)
  - متغیر رشته‌ای برای ذخیره خط خوانده شده
- حلقه while: تا زمانی که خط جدیدی برای خواندن وجود داشته باشد ادامه می‌یابد.

۴.۲.۵ کار با زمان

```
  #include <ctime>

  std::string get_current_time() {
    time_t now = time(nullptr);
    char buffer[20];
    strftime(buffer, sizeof(buffer),
    "%Y%m%d_%H%M%S", localtime(&now));
    return std::string(buffer);
  }
```

توضیح توابع زمان:

- time_t: نوع داده‌ای برای نگهداری زمان است که تعداد ثانیه‌ها از سال ۱۹۷۰ را ذخیره می‌کند.
- time(nullptr): زمان فعلی سیستم را برمی‌گرداند.
- localtime(): زمان را به فرمت قابل خواندن تبدیل می‌کند.
- strftime(): زمان را به رشته با فرمت دلخواه تبدیل می‌کند. پارامترهای آن:
  - بافر برای ذخیره نتیجه

- اندازه بافر
- الگوی فرمت (مثلاً %Y برای سال، %m برای ماه، و غیره)
- ساختار زمان

۳.۵ پروژه: پیاده‌سازی IDE ساده

در این بخش، با استفاده از مفاهیمی که تا اینجا آموخته‌ایم، یک IDE ساده خط فرمان پیاده‌سازی می‌کنیم. این IDE قابلیت‌های زیر را خواهد داشت:

- کامپایل برنامه‌های C++
- اجرای برنامه در پنجره فعلی یا پنجره جدید
- ذخیره و نمایش گزارش‌های کامپایل

۱.۳.۵ گام اول: تعریف ساختارها

ابتدا کتابخانه‌های مورد نیاز و ساختارهای داده را تعریف می‌کنیم:

```cpp
#include <iostream>
#include <string>
#include <cstdlib>
#include <ctime>
#include <fstream>
#include <vector>

// Store compilation results
struct CompilerResult {
  bool success;         // Status of compilation
  std::string output;   // Compiler messages
};

// Store user commands
struct Command {
  std::string name;    // Command name
  std::string args;    // Command arguments
};

// Store compilation logs
struct CompileLog {
  std::string timestamp;    // Time of compilation
  std::string filename;     // Source file name
  bool success;             // Compilation status
  std::string output;       // Compiler output
};
```

۲.۳.۵ گام دوم: توابع کمکی و پایه

سپس توابع کمکی مورد نیاز را پیاده‌سازی می‌کنیم.

۱. تابع ایجاد برچسب زمانی:

```
std::string get_timestamp() {
  time_t now = time(nullptr);
  char timestamp[20];
  strftime(timestamp, sizeof(timestamp),
  "%Y%m%d_%H%M%S", localtime(&now));
  return std::string(timestamp);
}
```

۲. تابع پردازش دستور:

```
Command parse_command(const std::string& input) {
  Command cmd;
  size_t space_pos = input.find(' ');

  if (space_pos != std::string::npos) {
    cmd.name = input.substr(0, space_pos);
    cmd.args = input.substr(space_pos + 1);
  } else {
    cmd.name = input;
  }

  return cmd;
}
```

۳.۳.۵ گام سوم: توابع اصلی برنامه

۱. تابع ذخیره گزارش:

```
bool save_compile_log(const CompileLog& log) {
  // Create logs directory if it doesn't exist
  system("if not exist logs mkdir logs");

  // Open log file in append mode
  std::ofstream log_file("logs/compile_log.txt",
  std::ios::app);

  if (!log_file.is_open()) {
    return false;
  }

```

```
    // Write log entry
    log_file << "\n=== Compilation Log ===\n"
    << "Time: " << log.timestamp << "\n"
    << "File: " << log.filename << "\n"
    << "Status: " << (log.success ? "Success" : "Failed")
    << "\n"
    << "Output:\n" << log.output << "\n"
    << "====================\n";

    log_file.close();
    return true;
  }
```

۲. تابع نمایش گزارش‌ها:

```
  void show_recent_logs(int count = 5) {
    std::ifstream log_file("logs/compile_log.txt");
    if (!log_file.is_open()) {
      std::cout << "No compilation logs found.\n";
      return;
    }

    std::vector<std::string> logs;
    std::string line;
    std::string current_log;

    // Read all logs
    while (std::getline(log_file, line)) {
      if (line.find("=== Compilation Log ===") !=
      std::string::npos) {
        if (!current_log.empty()) {
          logs.push_back(current_log);
        }
        current_log = line + "\n";
      } else {
        current_log += line + "\n";
      }
    }

    if (!current_log.empty()) {
      logs.push_back(current_log);
    }

    // Show the most recent logs
    int start = std::max(0,
    static_cast<int>(logs.size()) - count);
    for (int i = start; i < logs.size(); i++) {
```

```
      std::cout << logs[i] << std::endl;
    }

    log_file.close();
  }
```

۳. تابع کامپایل:

```
  CompilerResult compile_file(const std::string& filename) {
    CompilerResult result;
    CompileLog log;

    // Initialize log
    log.timestamp = get_timestamp();
    log.filename = filename;

    // Create compilation command
    std::string cmd = "g++ " + filename +
    " -o program.exe 2>&1";

    // Execute compiler and read output
    FILE* pipe = _popen(cmd.c_str(), "r");
    if (!pipe) {
      result.success = false;
      result.output = "Failed to execute compiler";
      log.success = false;
      log.output = result.output;
      save_compile_log(log);
      return result;
    }

    // Read output
    char buffer[128];
    while (fgets(buffer, sizeof(buffer), pipe)) {
      result.output += buffer;
    }
    _pclose(pipe);

    // Check compilation status
    result.success = (result.output.find("error") ==
    std::string::npos);

    // Save log
    log.success = result.success;
    log.output = result.output;
    save_compile_log(log);

```

```
  return result;
}
```

۴. تابع اجرا:

```
bool run_program(bool new_window = false) {
  std::string cmd;
  if (new_window) {
    cmd = "start cmd /c \"program.exe & pause\"";
  } else {
    cmd = "program.exe";
  }
  return (system(cmd.c_str()) == 0);
}
```

۴.۳.۵ گام چهارم: تابع اصلی برنامه

```
int main() {
  std::string input;
  bool running = true;

  // Print welcome message and help
  std::cout << "Simple IDE - Available commands:\n"
  << "  compile <filename> : Compile a file\n"
  << "  run                : Run in current window\n"
  << "  runw               : Run in new window\n"
  << "  logs [count]       : Show recent logs\n"
  << "  exit               : Exit IDE\n\n";

  while (running) {
    std::cout << "IDE> ";
    std::getline(std::cin, input);

    Command cmd = parse_command(input);

    if (cmd.name == "compile") {
      if (cmd.args.empty()) {
        std::cout << "Error: Please specify "
        << "a file to compile\n";
        continue;
      }

      CompilerResult result = compile_file(cmd.args);
```

```
        if (result.success) {
          std::cout << "Compilation successful!\n";
        } else {
          std::cout << "Compilation failed:\n"
          << result.output << std::endl;
        }
      }
      else if (cmd.name == "run") {
        if (!run_program(false)) {
          std::cout << "Failed to run program\n";
        }
      }
      else if (cmd.name == "runw") {
        if (!run_program(true)) {
          std::cout << "Failed to run program\n";
        }
      }
      else if (cmd.name == "logs") {
        int count = 5;  // Default count
        if (!cmd.args.empty()) {
          try {
            count = std::stoi(cmd.args);
          } catch (...) {
            std::cout << "Invalid count. Using "
            << "default (5)\n";
          }
        }
        show_recent_logs(count);
      }
      else if (cmd.name == "exit") {
        running = false;
      }
      else if (!cmd.name.empty()) {
        std::cout << "Unknown command. Type 'help' "
        << "for commands.\n";
      }
    }

    return 0;
  }
```