A black background with purple text

AI-generated content may be incorrect.

**FUNDEMANTALS OF ALGORITHM & COMPUTER PROBLEM SOLVING (CSC126)**

**SEMESTER MARCH-AUGUST 2024/2025**

PROJECT REPORT:

**Go-Kart Booking System**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Group Members Details | | | | |
|  | Group Members | Student No. | Group | Program |
| 1 | IRFAN SHAH BIN MAIZUL HISHAM | 2025171523 | RCDCS1101A | CDSC110 |
| 2 | SHAHRIN AREFF SHAH BIN SHAH RIZAL | 2025151503 | RCDCS1101A | CDSC110 |
| 3 | MUHAMMAD BIN ABDUL AZIZ | 2025507823 | RCDCS1101B | CDSC110 |
| Telephone Number (Leader): | | 0196093161 | | |

Lecturer’s Name: Nora Yanti Binti Che Jan

**Table of Contents**

[**1.0**  **PROJECT SUMMARY** 3](#_Toc203407030)

[**2.0**  **OBJECTIVES** 3](#_Toc203407031)

[**3.0**  **ALGORITHM DESIGN** 4](#_Toc203407032)

[**3.1**  **Flowcharts** 4](#_Toc203407033)

[**3.2**  **Pseudocode** 11](#_Toc203407034)

[**4.0**  **CODING GUIDELINES** 17](#_Toc203407035)

[**5.0**  **SOURCE CODE** 18](#_Toc203407036)

[**6.0**  **SAMPLES OF INPUT AND OUTPUT** 18](#_Toc203407037)

[**7.0**  **DISCUSSIONS** 19](#_Toc203407038)

[**8.0**  **CONLUSION** 19](#_Toc203407039)

[**9.0**  **PRESENTATION** 19](#_Toc203407040)

[**10.0**  **REFERENCES** 20](#_Toc203407041)

# **1.0 PROJECT SUMMARY**

In today’s modern world, people want to complete their work quickly and efficiently in order to save time and energy. The same concept applies to booking system including in Go-Kart’s business. By having an efficient booking system, it will ease both sides which are the staff and the customer and even can reduce human error. Therefore, this project aims to create a smooth and user-friendly booking system that applies every structure learned to show our understanding of the concept. As an overview, customer needs to input a few pieces of information such as the driver’s name, age and license availability. To make the coding easier, the system will ask whether the customer wants to play alone or by group before asking the information mentioned. For the group mode, we used array to make it easier for the system to store the data. The size of the array is based on total drivers entered by the customer. Then, the customer has to make few selections based on each function made in the project. Those functions including race formats, type of tracks, engine capacity, total laps, racing gears, and membership status. After all selections are completed, the system will calculate total price need to be paid after applied the membership discount if available. An output or receipt also will be displayed that shows the booking details. The system will be repeated from the start if there is a new customer. All functions are coded in C++ with few guidelines to standardise our code.

# **2.0 OBJECTIVES**

1. To implement and showcase every controlled structure learned in CSC126 suitably based on the problem.
2. To create a system which trakcs and allows for customers to do booking.
3. To calculate the cost of booking accordingly based on factors such as driver count and engine capacity

# **3.0 ALGORITHM DESIGN**

[Insert algorithm design explainantion here]

## **3.1 Flowcharts**

|  |
| --- |
| A screenshot of a computer screen  AI-generated content may be incorrect.  *Figure 3.1.0 Main Function* |

|  |
| --- |
| A screenshot of a computer screen  AI-generated content may be incorrect.  *Figure 3.1.1 Menu Function* |

|  |  |
| --- | --- |
| A screenshot of a video game  AI-generated content may be incorrect.  *Figure 3.1.2 displayMenu Function* | A screenshot of a computer  AI-generated content may be incorrect.  *Figure 3.1.3 displayRaceFormat Function* |

|  |  |
| --- | --- |
| A green rectangles with white text  AI-generated content may be incorrect.  *Figure 3.1.4* | A screenshot of a computer screen  AI-generated content may be incorrect.  *Figure 3.1.5* |

|  |
| --- |
| A screenshot of a computer program  AI-generated content may be incorrect.  Figure 3.1.6 |

**Input Validation Functions:**

|  |
| --- |
| A screenshot of a computer program  AI-generated content may be incorrect.  *Figure 3.1.7 getValidIntegerInput Function* |

|  |
| --- |
| A screenshot of a computer screen  AI-generated content may be incorrect.  *Figure 3.1.8 getValidCharacterInput Function* |

|  |
| --- |
| A screenshot of a computer  AI-generated content may be incorrect.  *Figure 3.1.9 getBookingType Function* |

|  |
| --- |
| A screenshot of a computer screen  AI-generated content may be incorrect.  Figure 3.1.10 |

|  |
| --- |
| A screenshot of a computer  AI-generated content may be incorrect.  Figure 3.1.11 |

|  |
| --- |
| A screenshot of a computer screen  AI-generated content may be incorrect.  Figure 3.1.12 |

|  |
| --- |
| A screenshot of a computer screen  AI-generated content may be incorrect.  Figure 3.1.13 |

|  |
| --- |
| A screenshot of a computer screen  AI-generated content may be incorrect.  Figure 3.1.14 |

|  |
| --- |
| A screenshot of a computer screen  AI-generated content may be incorrect.  Figure 3.1.15 |

|  |
| --- |
| A screenshot of a computer  AI-generated content may be incorrect.  Figure 3.1.16 |

|  |  |
| --- | --- |
| A screenshot of a computer  AI-generated content may be incorrect.  Figure 3.1.17 | A screenshot of a computer screen  AI-generated content may be incorrect.  Figure 3.1.18 |

|  |
| --- |
| A screenshot of a computer screen  AI-generated content may be incorrect.  Figure 3.1.19 |

|  |
| --- |
| A screenshot of a computer  AI-generated content may be incorrect.  Figure 3.1.20 |

|  |
| --- |
| A screenshot of a computer  AI-generated content may be incorrect.  Figure 3.1.21 |

|  |
| --- |
| A screenshot of a computer program  AI-generated content may be incorrect.  Figure 3.1.22 |

## **3.2 Pseudocode**

|  |
| --- |
| FUNCTION INTEGER main  DECLARE CHARACTER continueChoice  DO  CALL menu()  //Initializations (in order)  STRING bookingType  DECLARE INTEGER driverCount  DECLARE STRING driverName[5]  DECLARE INTEGER driverAge[5]  DECLARE CHARACTER license[5]  DECLARE STRING raceFormat  DECLARE STRING track    DECLARE INTEGER engineCapacity[5]  DECLARE INTEGER laps  DECLARE STRING helmetSize[5], suitSize[5]  DECLARE INTEGER shoeSize[5]  DECLARE FLOAT helmetPrice[5], suitPrice[5], shoePrice[5]  DECLARE FLOAT gearPrice[5]  // The Crux of the Go-Kart Booking System  SET bookingType = getBookingType()    SET driverCount = getDriverCount(bookingType)  CALL getDriverDetails(driverCount, driverName, driverAge, license)  SET raceFormat = setRaceFormat(bookingType)  SET track = setTrack(bookingType, raceFormat)  SET setEngineCapacity(driverCount, driverName, driverAge, license, engineCapacity)  SET laps = setLaps(driverCount, raceFormat)  FOR INTEGER i = 0; i < driverCount; i++  DISPLAY "Driver: ", driverName[i]  selectHelmet(i,helmetSize, helmetPrice)  selectSuit(i,suitSize, suitPrice)  selectShoe(i, shoeSize, shoePrice)  END FOR  DECLARE FLOAT totalGearPrice  SET totalGearPrice = calculateTotalGearPrice(driverCount, helmetPrice, suitPrice, shoePrice, gearPrice)  DECLARE FLOAT membershipDiscount  SET membershipDiscount = setMembershipDiscount()  DECLARE FLOAT totalPrice  SET totalPrice = calculatePrice(engineCapacity, laps, driverCount, totalGearPrice, membershipDiscount)  OUTPUT "Race Format: ", raceFormat  OUTPUT "Track: ", track  DISPLAY "Race Format: ", raceFormat  DISPLAY "Track: ", track  FOR INTEGER i = 0; i < driverCount; i++  OUTPUT "Driver: ", driverName[i]  OUTPUT "Age: ", driverAge[i]  OUTPUT "Has License: ", license[i]  OUTPUT "Engine Capacity: ", engineCapacity[i] << "cc"  OUTPUT "Price per CC: RM", PRICEPERCC, "/cc"  OUTPUT "Go-Kart Price: RM ", PRICEPERCC \* engineCapacity[i]  OUTPUT "Helmet Size: ", helmetSize[i]  OUTPUT "Helmet Price: RM ", helmetPrice[i]  OUTPUT "Suit Size: ", suitSize[i]  OUTPUT "Suit Price: RM ", suitPrice[i]  OUTPUT "Shoe Size: ", shoeSize[i], "cm"  OUTPUT "Shoe Price: RM ", shoePrice[i]  OUTPUT "Gear Price: RM ", gearPrice[i]  END FOR    OUTPUT "Total Gear Price: RM", totalGearPrice  OUTPUT "Membership Discount\t: ", membershipDiscount \* 100, "%"  OUTPUT "Total : RM", std::setprecision(2), totalPrice  counter.totalIncome += totalPrice  counter.customer += driverCount;  PROMPT "Continue for another customer? (Y/N): "  SET continueChoice = getValidCharacterInput()  WHILE std::toupper(continueChoice) == 'Y'  OUTPUT "Total Number Drivers: ", counter.customer  OUTPUT "Total Income: RM ", counter.totalIncome  OUTPUT "Total Number of 120cc Go-Kart Booked: ", counter.cc120  OUTPUT "Total Number of 200cc Go-Kart Booked: ", counter.cc200  OUTPUT "Total Number of 270cc Go-Kart Booked: ", counter.cc270  OUTPUT "Total Number of members: ", counter.membership  OUTPUT "Total Number of Section 9 Circuit booking: ", counter.section9  OUTPUT "Total Number of Blackrock Circuit booking: ", counter.blackrock  OUTPUT "Total Number of Rushline Dash booking: ", counter.rushlineDash  OUTPUT "Total Number of Chrono Pass booking: ", counter.choroPass  OUTPUT "Total Number of Torque Strip booking: ", counter.torqueStrip  RETURN 0  END FUNCTION  MODULE menu()  DECLARE ENUMERATION menu {start = 1, format = 2, track = 3, gear = 4, engine = 5}    DECLARE menu menuOption  DO  CALL displayMenu()  PROMPT "Choose your option: "  SET menuOption = static\_cast<menu>(getValidIntegerInput("option", 1, 5)); // Converts integers into menu enums  IF menuOption == format THEN  Option displayRaceFormat("Group")  END IF  IF menuOption == track THEN  Option displayTrackList()  END IF  IF menuOption == gear THEN  Option displayRacingGear()  END IF  IF menuOption == engine THEN  displayEngineCapacities(18, 'Y')  END IF  WHILE menuOption != start  END MODULE  MODULE displayMenu()  DISPLAY "Go-Kart Booking System"  DISPLAY "1 - Start Booking"  DISPLAY "2 - Race Formats"  DISPLAY "3 - Track Lists"  DISPLAY "4 - Racing Gear Sizes & Prices"  DISPLAY "5 - Go-Kart Engine Capacities"  END MODULE  MODULE displayRaceFormat(STRING bookingType)  DISPLAY "Available Race Formats:"  DISPLAY "1 - Circuit Race"  DISPLAY "2 - Sprint Race"  DISPLAY "3 - Time Trial"  DISPLAY "4 - Drag Race"  IF bookingType == "Group" THEN  DISPLAY "5 - Eliminationn Race (Group)"  END IF  END MODULE  MODULE displayTrackList()  DISPLAY "Available Tracks:"  DISPLAY "1 - Section 9 Circuit"  DISPLAY "2 - Chrono Pass"  DISPLAY "3 - Rushline Dash"  DISPLAY "4 - Blackrock Circuit"  DISPLAY "5 - Torque Strip"  END MODULE  MODULE displayRacingGear()  DISPLAY "Racing Gears:"  DISPLAY " Helmets Suits Shoes"  DISPLAY "Size S: RM 2 RM 5 Size 20-25 cm: RM 3"  DISPLAY "Size M: RM 4 RM 10 Size 26-35 cm: RM 6"  DISPLAY "Size L: RM 6 RM 15"  DISPLAY "Size XL RM 8 RM 20"  END MODULE  MODULE displayEngineCapacities(INTEGER age, CHARACTER license)  DISPLAY "Available Engine Capacities:"  DISPLAY "1 - 100cc (Under 13)"  IF age >= 18 THEN  DISPLAY "2 - 200cc"  IF (license == 'Y') THEN  DISPLAY "3 - 270cc (Requires License)"  END IF  END MODULE  FUNCTION INTEGER getValidIntegerInput(STRING inputType, INTEGER minValue, INTEGER maxValue)  int input;  READ input;  WHILE std::cin.fail() || (input < minValue || input > maxValue)  PROMPT "\t\t\tPlease enter a valid " << inputType << " (" << minValue << " - " << maxValue << "): "  READ input  END WHILE  RETURN input  END FUNCTION  FUNCTION CHARACTER getValidCharacterInput()  STRING input  READ input  WHILE input.length() != 1 || !std::isalpha(input[0])  PROMPT "Please enter a valid input (Y/N): "  READ input  END WHILE  RETURN std::toupper(input[0])  END FUNCTION  FUNCTION STRING getBookingType()  DECLARE INTEGER bookingTypeID  DISPLAY "1 - Solo"  DISPLAY "2 - Group (Maximum 5)"  PROMPT "Please choose your desired booking (1 - 2): "  SET bookingTypeID = getValidIntegerInput("booking type", 1, 2)  IF bookingTypeID == 1 THEN  SET counter.solo++  RETURN "Solo"  ELSE  SET counter.group++  RETURN "Group"  END IF  END FUNCTION  FUNCTION INTEGER getDriverCount(STRING& bookingType)  IF bookingType == "Solo" THEN  RETURN 1  END IF  PROMPT "Please enter the number of drivers (2 - 5): "  RETURN getValidIntegerInput("number of drivers", 2, 5)  END FUNCTION  MODULE getDriverDetails(INTEGER& driverCount, STRING driverName[], INTEGER driverAge[], CHARACTER license[])  FOR int i = 0; i < driverCount; i++  PROMPT "Driver #", (i+1), " Name: "  READ driverName[i]  PROMPT "Driver #", (i+1), " Age: "  READ driverAge[i]  PROMPT "Does Driver #", (i+1), " has a license? (Y/N): "  SET license[i] = getValidCharacterInput()  END FOR  END MODULE  FUNCTION STRING setRaceFormat(STRING& bookingType)  DECLARE INTEGER raceFormatID  DECLARE STRING raceFormat  CALL displayRaceFormat(bookingType)  IF bookingType == "Solo" THEN  PROMPT "Please choose the race format (1 - 4): "  SET raceFormatID = getValidIntegerInput("race format", 1, 4);  ELSE IF bookingType == "Group" THEN  PROMPT "Please choose the race format (1 - 5): "  SET raceFormatID = getValidIntegerInput("race format", 1, 5)  END IF  SWITCH (raceFormatID)  CASE 1: RETURN "Circuit Race";  CASE 2: RETURN "Sprint Race";  CASE 3: RETURN "Time Trial";  CASE 4: RETURN "Drag Race";  CASE 5: RETURN "Elimination Race";  DEFAULT: RETURN "Invalid Race Format";  END FUNCTION  FUNCTION STRING setTrack(STRING& bookingType, STRING& raceFormat)  DECLARE INTEGER trackID  IF bookingType == "Group" AND raceFormat == "Circuit Race" THEN  DISPLAY "Available Track:"  DISPLAY "1 - Section 9 Circuit"  DISPLAY "2 - Blackrock Circuit"  PROMPT "Please choose your track (1 - 2): "  SET trackID = getValidIntegerInput("track, 1, 2)  ELSE IF raceFormat == "Time Trial" THEN  DISPLAY "Available Track:"  DISPLAY "1 - Section 9 Circuit"  DISPLAY "2 - Blackrock Circuit"  DISPLAY "3 - Rushline Dash"  DISPLAY "4 - Chrono Pass"  PROMPT "Please choose your track (1 - 4): "  SET trackID = getValidIntegerInput("track, 1, 4)  END IF  SWITCH (trackID)  CASE 1: counter.section9++; RETURN "Section 9 Circuit";  CASE 2: counter.blackrock++; RETURN "Blackrock Circuit";  CASE 3: counter.rushlineDash++; RETURN "Rushline Dash";  DEFAULT: counter.choroPass++; RETURN "Chrono Pass";  IF bookingType == "Solo" AND raceFormat == "Circuit Race" THEN  DISPLAY "Available Track: Section 9 Circuit"  DISPLAY "Defaulting to said track"  SET counter.section9++  RETURN "Section 9 Circuit"  ELSE IF raceFormat == "Elimination Race" THEN  DISPLAY "Available Track: Blackrock Circuit"  DISPLAY "Defaulting to said track"  SET counter.blackrock++  RETURN "Blackrock Circuit"  ELSE IF raceFormat == "Sprint Race" THEN  DISPLAY "Available Track: Rushline Dash"  DISPLAY "Defaulting to said track"  SET counter.rushlineDash++  RETURN "Rushline Dash"  ELSE IF raceFormat == "Drag Race" THEN  DISPLAY "Available Track: Torque Strip"  DISPLAY "Defaulting to said track"  SET counter.torqueStrip++  RETURN "Torque Strip"  END IF  RETURN "Track"  END FUNCTION    MODULE setEngineCapacity(INTEGER driverCount, STRING driverName[], INTEGER driverAge[], CHARACTER license[])  FOR INTEGER i = 0, i < driverCount, i++  DISPLAY "Driver : ", driverName[i]  CALL displayEngineCapacities(driverAge[i], license[i])  IF driverAge[i] < 13 THEN  DISPLAY "Driver under 13: Go-Kart must be under 200cc"  DISPLAY "Defaulting to 120cc"  SET engineCapacity[i] = 120  SET counter.cc120++  CONTINUE  END IF  IF license[i] != 'Y' THEN  DISPLAY "Choose your desired engine capacity (1 - 2): "  SET engineCapacity[i] = getValidIntegerInput("engine capacity", 1, 2)  END IF  IF license[i] == 'Y'  DISPLAY "Choose your desired engine capacity (1 - 3): "  SET engineCapacity[i] = getValidIntegerInput("engine capacity", 1, 3)  END IF  SWITCH (engineCapacity[i])  CASE 1: engineCapacity[i] = 120, SET counter.cc120++, BREAK  CASE 2: engineCapacity[i] = 200, SET counter.cc200++, BREAK  DEFAULT: engineCapacity[i] = 270, SET counter.cc270++, BREAK  END FOR  END MODULE  FUNCTION setLaps(INTEGER driverCount, STRING& raceFormat)  DECLARE INTEGER laps  IF raceFormat == "Circuit Race" THEN  DISPLAY "Available number of laps: 2, 3, 4"  PROMPT "How many laps would you like? (2/3/4): "  READ laps  ELSE IF raceFormat == "Sprint Race" OR raceFormat == "Drag Race" THEN  DISPLAY "Sprint Races and Drag Races only have 1 lap"  SET laps = 1  ELSE IF raceFormat == "Time Trial" THEN  DISPLAY "Available number of laps: Unlimited"  PROMPT "How many laps would you like? :"  READ laps  ELSE  DISPLAY "Number of laps corresponds with the number of drivers"  SET laps = driverCount - 1  DISPLAY "Laps = Driver Count - 1"  DISPLAY "Laps = ", driverCount, " - 1"  DISPLAY "Laps = ", laps  END IF  RETURN LAPS  END FUNCTION  FUNCTION float selectHelmet (INTEGER i, STRING helmetSize[], FLOAT helmetPrice[])  PROMPT "Please choose your helmet size (S/M/L/XL): "  SET helmetSize[i] = getValidSizeInput("helmet size")  IF helmetSize[i] == "S" THEN  helmetPrice[i] = 2  END IF  IF helmetSize[i] == "M" THEN  helmetPrice[i] = 4  END IF  IF helmetSize[i] == "L" THEN  helmetPrice[i] = 6  END IF  IF helmetSize[i] == "XL" THEN  helmetPrice[i] = 8  END IF  RETURN helmetPrice[i]  END FUNCTION  FUNCTION float selectSuit (INTEGER i, STRING suitSize[], FLOAT suitPrice[])  PROMPT "Please choose your suit size (S/M/L/XL): "  SET suitSize[i] = getValidSizeInput("suit size")    IF suitSize[i] == "S" THEN  suitPrice[i] = 2  END IF  IF suitSize[i] == "M" THEN  suitPrice[i] = 4  END IF  IF suitSize[i] == "L" THEN  suitPrice[i] = 6  END IF  IF helmetSize[i] == "XL" THEN  suitPrice[i] = 8  END IF  RETURN suitPrice[i]  END FUNCTION  FUNCTION FLOAT selectShoe(INTEGER i, INTEGER shoeSize[], FLOAT shoePrice[])  PROMPT "Please choose your shoe size (20 - 35): "  READ shoeSize[i]  WHILE std::cin.fail() || (shoeSize[i] < 20 || shoeSizep[i] > 35)  PROMPT "Please choose a valid shoe size (20 - 35): "  READ shoeSize[i]  END WHILE  IF shoeSize[i] <= 25 THEN  shoePrice[i] = 3  ELSE  shoePrice[i] = 6  END IF  RETURN shoePrice[i]  END FUNCTION  FUNCTION FLOAT calculateTotalGearPrice(INTEGER& driverCount, FLOAT helmetPrice[], FLOAT suitPrice[], FLOAT gearPrice[])  DECLARE totalGearPrice  SET totalGearPrice = 0  FOR INTEGER i = 0, i < driverCount, i++  gearPrice[i] = helmetPrice[i] + suitPrice[i] + shoePrice [i]  totalGearPrice += gearPrice[i]  END FOR  RETURN totalGearPrice  END FUNCTION  FUNCTION FLOAT setMembershipDiscount()  DECLARE CHAR membership  PROMPT "Do you have a membership? (Y/N): "  SET membership = getValidCharacterInput()  IF membership == 'Y' THEN  SET counter.membership++  RETURN 0.1  ELSE  RETURN 0.0  END IF  END FUNCTION  FUNCTION FLOAT calculatePrice(int engineCapacity[], int laps, int driverCount, float totalGearPrice, float membershipDiscount)  SET pricePerCC = 03  SET totalKartPrice = 0  FOR i = 0, i < driverCount, i++  totalKartPrice += (engineCapacity[i] \* pricePerCC) \* laps  END FOR    finalPrice = (totalKartPrice + totalGearPrice) \* (1 - membershipDiscount)  END FUNCTION |

# **4.0 CODING GUIDELINES**

1. cameCase naming convention

This is a popular naming convention and it is used by our lecturer to teach C++.

1. Tab Width: 8

The usage of 8 tab width is to easily spot excessive nesting easily which ties to the next point.

1. Maximum Level of Indentation: 3

To quote Linus Torvalds on the Linux Kernal coding guideline: “If you need more than 3 levels of indentation, your code is broken anyway and should fix it”.

1. Indentation Style: Stroustrup

Bajrne Stroustrup is the inventor of C++ and his style of coding is in the C++ International Standard Organization (ISO).

1. No *using namespace std*

The usage of *using namespace std* in simpler programs is okay and sometimes advised to but at higher levels of programming, the usage of it can cause problems. To avoid the compiler being confused and to better make the usage of functions from the standard library clearer, *using namesapce std* is omitted from our code.

1. No abbreviations

Abbreviations can make coding easier, but with the advent of IDE’s autocomplete, coding takes less keystroke than ever. Thus, to make the code readable to everyone, it does not matter if you’re working on it or just an outsider, the code will still be comprehensible even if you have no idea on Go-Kart or how this booking works.

# **5.0 SOURCE CODE**

|  |
| --- |
| // CSC126 Group Project  // Go-Kart Booking System  // IRFAN SHAH BIN MAIZUL HISHAM         (2025171523) (shahxvi)  // SHAHRIN AREFF SHAH BIN SHAH RIZAL    (2025151503) (rinnnnnn17)  // MUHAMMAD BIN ABDUL AZIZ              (2025507823) (Kundoo)  #include <iostream>  #include <string>  #include <cmath>  #include <iomanip>  #include <cctype>  #include <algorithm>  #define PRICEPERCC 0.3  using std::string;  using std::cout;  using std::endl;  using std::cin;  void menu();  void displayMenu();  void displayRaceFormat(string bookingType);  void displayTrackList();  void displayRacingGear();  void displayEngineCapacities(int age, char license);  int getValidIntegerInput (string inputType, int minValue, int maxValue);  char getValidCharacterInput();  string getBookingType();  int getDriverCount(string& bookingType);  void getDriverDetails(int& driverCount, string driverName[], int driverAge[], char license[]);  string setRaceFormat(string& bookingType);  string setTrack(string& bookingType, string& raceFormat);  void setEngineCapacity(int driverCount, string driverName[], int driverAge[], char license[], int engineCapacity[]);  int setLaps(int driverCount, string& raceFormat);  string getValidSizeInput(string inputType);  float selectHelmet(int i, string helmetSize[], float helmetPrice[]);  float selectSuit(int i, string suitSize[], float suitPrice[]);  float selectShoe(int i, int shoeSize[], float shoePrice[]);  float calculateTotalGearPrice(int& driverCount, float helmetPrice[], float suitPrice[], float shoePrice[], float gearPrice[]);  float setMembershipDiscount();  float calculatePrice(int engineCapacity[], int laps, int driverCount, float gearPrice, float membershipDiscount);  struct Counters {          int customer = 0;          float totalIncome = 0;          int solo = 0;          int group = 0;          int membership = 0;          int cc120 = 0;          int cc200 = 0;          int cc270 = 0;          int section9 = 0;          int blackrock = 0;          int rushlineDash = 0;          int choroPass = 0;          int torqueStrip = 0;  };  Counters counter;  int main()  {          char continueChoice;          do {                  menu();                  //Initializations (in order)                  string bookingType;                  int driverCount;                  string driverName[5];                  int driverAge[5];                  char license[5];                  string raceFormat;                  string track;                    int engineCapacity[5];                  int laps;                  string helmetSize[5], suitSize[5]; int shoeSize[5];                  float helmetPrice[5], suitPrice[5], shoePrice[5];                  float gearPrice[5];                  // The Crux of the Go-Kart Booking System                  bookingType = getBookingType();                    driverCount = getDriverCount(bookingType);                  getDriverDetails(driverCount, driverName, driverAge, license);                  system("cls");                  raceFormat = setRaceFormat(bookingType);                  track = setTrack(bookingType, raceFormat);                  setEngineCapacity(driverCount, driverName, driverAge, license, engineCapacity);                  laps = setLaps(driverCount, raceFormat);                    for (int i = 0; i < driverCount; i++) {                          system("cls");                          displayRacingGear();                          cout << "\n\t\t\tDriver: " << driverName[i];                          selectHelmet(i,helmetSize, helmetPrice);                          selectSuit(i,suitSize, suitPrice);                          selectShoe(i, shoeSize, shoePrice);                  }                  float totalGearPrice = calculateTotalGearPrice(driverCount, helmetPrice, suitPrice, shoePrice, gearPrice);                  float membershipDiscount = setMembershipDiscount();                  float totalPrice = calculatePrice(engineCapacity, laps, driverCount, totalGearPrice, membershipDiscount);                  system("cls");                  // Summary for each customers                  cout << "\n\t\t\tRace Format: " << raceFormat                       << "\n\t\t\tTrack: " << track << "\n";                  std::cout << std::fixed << std::showpoint << std::setprecision(2);                  for (int i = 0; i < driverCount; i++) {                          cout << "\n\t\t\tDriver\t\t\t: "        << driverName[i]                               << "\n\t\t\tAge\t\t\t: "           << driverAge[i]                               << "\n\t\t\tHas License\t\t: "     << license[i]                               << "\n\t\t\tEngine Capacity\t\t: " << engineCapacity[i] << "cc"                               << "\n\t\t\tPrice per CC\t\t: "    << "RM " << PRICEPERCC << "/cc"                               << "\n\t\t\tGo-Kart Price\t\t: RM " << PRICEPERCC \* engineCapacity[i]                               << "\n\t\t\tHelmet Size\t\t: "     << helmetSize[i]                               << "\n\t\t\tHelmet Price\t\t: RM " << helmetPrice[i]                               << "\n\t\t\tSuit Size\t\t: "       << suitSize[i]                               << "\n\t\t\tSuit Price\t\t: RM "   << suitPrice[i]                               << "\n\t\t\tShoe Size\t\t: "       << shoeSize[i] << "cm"                               << "\n\t\t\tShoe Price\t\t: RM "   << shoePrice[i]                               << "\n\t\t\tGear Price\t\t: RM "   << gearPrice[i] << "\n";                  }                  cout << "\n\t\t\tTotal Gear Price\t: RM " << totalGearPrice                            << "\n\t\t\tMembership Discount\t: " << membershipDiscount \* 100 << "%"                            << "\n\t\t\tTotal : RM " << totalPrice;                  counter.totalIncome += totalPrice;                  counter.customer += driverCount;                  cout << "\n\t\t\tContinue for another customer? (Y/N): ";                  continueChoice = getValidCharacterInput();                  system("cls");          } while (std::toupper(continueChoice) == 'Y');          // Overall Summary          cout << "\n\t\t\tTotal Number Drivers: " << counter.customer;          cout << "\n\t\t\tTotal Income: RM " << counter.totalIncome << endl;          cout << "\n\t\t\tTotal Number of 120cc Go-Kart Booked: " << counter.cc120;          cout << "\n\t\t\tTotal Number of 200cc Go-Kart Booked: " << counter.cc200;          cout << "\n\t\t\tTotal Number of 270cc Go-Kart Booked: " << counter.cc270 << endl;          cout << "\n\t\t\tTotal Number of members: " << counter.membership <<  endl;          cout << "\n\t\t\tTotal Number of Section 9 Circuit booking: " << counter.section9;          cout << "\n\t\t\tTotal Number of Blackrock Circuit booking: " << counter.blackrock;          cout << "\n\t\t\tTotal Number of Rushline Dash booking: " << counter.rushlineDash;          cout << "\n\t\t\tTotal Number of Chrono Pass booking: " << counter.choroPass;          cout << "\n\t\t\tTotal Number of Torque Strip booking: " << counter.torqueStrip;          cout << endl;          cout << "\n\t\t\t" << system("pause");          return 0;  }  void menu()  {          enum menu {start = 1, format = 2, track = 3, gear = 4, engine = 5};          menu menuOption;          do {                  displayMenu();                  cout << "\n\t\t\tChoose your option: ";                  menuOption = static\_cast<menu>(getValidIntegerInput("option", 1, 5)); // Converts integers into menu enums                  if (menuOption == format) {                          system("cls");                          displayRaceFormat("Group");             // Parameters to allow the menu to show every option                          system("pause");                          system("cls");                  }                  if (menuOption == track) {                          system("cls");                          displayTrackList();                          system("pause");                          system("cls");                  }                  if (menuOption == gear) {                          system("cls");                          displayRacingGear();                          system("pause");                          system("cls");                  }                  if (menuOption == engine) {                          system("cls");                          displayEngineCapacities(18, 'Y');       // Parameters to allow the menu to show every option                          system("pause");                          system("cls");                  }          } while (menuOption != start);  }  void displayMenu()  {          cout << "\t\_\_\_\_\_\_\_\_\_            \_\_\_\_\_\_ \_\_             \_\_\_\_\_  \_\_\_\_\_\_\_\_            \_\_\_\_\_\_ \_\_\_\_\_                 \_\_\_\_\_\_\_\_              \_\_\_\_\_                 "               << "\n\t\_\_  \_\_\_\_/\_\_\_\_\_       \_\_\_  //\_/\_\_\_\_\_ \_\_\_\_\_\_\_\_\_  /\_ \_\_\_  \_\_ )\_\_\_\_\_\_\_\_\_\_\_\_\_\_  /\_\_\_\_(\_)\_\_\_\_\_\_\_\_\_\_\_\_\_ \_ \_\_  \_\_\_/\_\_\_\_  \_\_\_\_\_\_\_\_\_\_  /\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_ "               << "\n\t\_  / \_\_ \_  \_\_ \\\_\_\_\_\_\_\_\_  ,<  \_  \_\_ `/\_  \_\_\_/  \_\_/ \_\_  \_\_  |  \_\_ \\  \_\_ \\\_  //\_/\_  /\_\_  \_\_ \\\_  \_\_ `/ \_\_\_\_\_ \\\_\_  / / /\_  \_\_\_/  \_\_/  \_ \\\_  \_\_ `\_\_ \\"               << "\n\t/ /\_/ / / /\_/ //\_\_\_\_\_/  /| | / /\_/ /\_  /   / /\_   \_  /\_/ // /\_/ / /\_/ /  ,<  \_  / \_  / / /  /\_/ /  \_\_\_\_/ /\_  /\_/ /\_(\_\_  )/ /\_ /  \_\_/  / / / / /"               << "\n\t\\\_\_\_\_/  \\\_\_\_\_/       /\_/ |\_| \\\_\_,\_/ /\_/    \\\_\_/   /\_\_\_\_\_/ \\\_\_\_\_/\\\_\_\_\_//\_/|\_| /\_/  /\_/ /\_/\_\\\_\_, /   /\_\_\_\_/ \_\\\_\_, / /\_\_\_\_/ \\\_\_/ \\\_\_\_//\_/ /\_/ /\_/ "               << "\n\t                                                                                         /\_\_\_\_/           /\_\_\_\_/                               "               << "\n\t\t\t1 - Start Booking"               << "\n\t\t\t2 - Race Formats"               << "\n\t\t\t3 - Track Lists"               << "\n\t\t\t4 - Racing Gear Sizes & Prices"               << "\n\t\t\t5 - Go-Kart Engine Capacities\n";  }  void displayRaceFormat(string bookingType)  {          cout << "\n\t\t\tAvailable Race Formats:"               << "\n\t\t\t1 - Circuit Race"               << "\n\t\t\t2 - Sprint Race"               << "\n\t\t\t3 - Time Trial"               << "\n\t\t\t4 - Drag Race";          if (bookingType == "Group") {                  cout << "\n\t\t\t5 - Eliminationn Race (Group)\n";          }  }  void displayTrackList()  {          cout << "\n\t\t\tAvailable Tracks:"                    << "\n\t\t\t1 - Section 9 Circuit"                    << "\n\t\t\t2 - Chrono Pass"                    << "\n\t\t\t3 - Rushline Dash"                    << "\n\t\t\t4 - Blackrock Circuit"                    << "\n\t\t\t5 - Torque Strip\n";  }  void displayRacingGear()  {          cout << "\n\t\t\tRacing Gears:"                            << "\n\t\t\t\t\t\tHelmets\t\t\tSuits\t\t\tShoes\n"                            << "\n\t\t\tSize S:\t\t\tRM 2\t\t\tRM 5\t\t\tSize 20-25 cm: RM 3"                            << "\n\t\t\tSize M:\t\t\tRM 4\t\t\tRM 10\t\t\tSize 26-35 cm: RM 6"                            << "\n\t\t\tSize L:\t\t\tRM 6\t\t\tRM 15"                            << "\n\t\t\tSize XL\t\t\tRM 8\t\t\tRM 20\n";  }  void displayEngineCapacities(int age, char license)  {          cout << "\n\t\t\tAvailable Engine Capacities:";          cout << "\n\t\t\t1 - 100cc (Under 13)";          if (age >= 18)                  cout << "\n\t\t\t2 - 200cc";          if (license == 'Y')                  cout << "\n\t\t\t3 - 270cc (Requires License)\n";  }  int getValidIntegerInput(string inputType, int minValue, int maxValue)  {          int input;          cin >> input;          while (cin.fail() || (input < minValue || input > maxValue)) {                  cin.clear();                  cin.ignore(1000, '\n');                  cout << "\t\t\tPlease enter a valid " << inputType << " (" << minValue << " - " << maxValue << "): ";                  cin >> input;          }          return input;  }  char getValidCharacterInput()  {          string input;          cin >> input;          while (input.length() != 1 || !std::isalpha(input[0])) {                  cin.clear();                  cin.ignore(1000, '\n');                  cout << "\t\t\tPlease enter a valid input (Y/N): ";                  cin >> input;          }          return std::toupper(input[0]);  }  string getBookingType()  {          int bookingTypeID;          cout << "\n\t\t\t1 - Solo\n"               << "\t\t\t2 - Group (Maximum 5)\n"               << "\t\t\tPlease choose your desired booking (1 - 2): ";          bookingTypeID = getValidIntegerInput("booking type", 1, 2);          if (bookingTypeID == 1) {                  counter.solo++;                  return "Solo";          }          else {                  counter.group++;                  return "Group";          }  }  int getDriverCount(string& bookingType)  {          if (bookingType == "Solo")                  return 1;          cout << "\t\t\tPlease enter the number of drivers (2 - 5): ";          return getValidIntegerInput("number of drivers", 2, 5);  }  void getDriverDetails(int& driverCount, string driverName[], int driverAge[], char license[])  {          for (int i = 0; i < driverCount; i++) {                  system("cls");                  cin.ignore();                  cout << "\n\t\t\tDriver #" << (i+1) << " Name: ";                  std::getline(cin, driverName[i]);                  cout << "\t\t\tDriver #" << (i+1) << " Age: ";                  cin >> driverAge[i];                  cout << "\t\t\tDoes Driver #" << (i+1) << " has a license? (Y/N): ";                  license[i] = getValidCharacterInput();          }  }  string setRaceFormat(string& bookingType)  {          int raceFormatID;          string raceFormat;          displayRaceFormat(bookingType);          if (bookingType == "Solo") {                  cout << "\n\t\t\tPlease choose the race format (1 - 4): ";                  raceFormatID = getValidIntegerInput("race format", 1, 4);          }          else if (bookingType == "Group") {                  cout << "\n\t\t\tPlease choose the race format (1 - 5): ";                  raceFormatID = getValidIntegerInput("race format", 1, 5);          }            switch (raceFormatID) {                  case 1: return "Circuit Race";                  case 2: return "Sprint Race";                  case 3: return "Time Trial";                  case 4: return "Drag Race";                  case 5: return "Elimination Race";                  default: return "Invalid Race Format";          }  }  string setTrack(string& bookingType, string& raceFormat)  {          int trackID;          if (bookingType == "Group" && raceFormat == "Circuit Race") {                  cout << "\n\t\t\tAvailable Track:"                            << "\n\t\t\t1 - Section 9 Circuit"                            << "\n\t\t\t2 - Blackrock Circuit"                            << "\n\t\t\tPlease choose your track (1 - 2): ";                  trackID = getValidIntegerInput("track", 1, 2);          }          else if (raceFormat == "Time Trial") {                  cout << "\n\t\t\tAvailable Track:"                            << "\n\t\t\t1 - Section 9 Circuit"                            << "\n\t\t\t2 - Blackrock Circuit"                            << "\n\t\t\t3 - Rushline Dash"                            << "\n\t\t\t4 - Chrono Pass"                            << "\n\t\t\tPlease choose your track (1 - 4): ";                  trackID = getValidIntegerInput("track", 1, 4);          }          switch (trackID) {                  case 1: counter.section9++; return "Section 9 Circuit";                  case 2: counter.blackrock++; return "Blackrock Circuit";                  case 3: counter.rushlineDash++; return "Rushline Dash";                  default: counter.choroPass++; return "Chrono Pass";          }          if (bookingType == "Solo" && raceFormat == "Circuit Race") {                  cout << "\n\t\t\tAvailable Track: Section 9 Circuit"                       << "\n\t\t\tDefaulting to said track\n";                  counter.section9++;                  return "Section 9 Circuit";          }          else if (raceFormat == "Elimination Race") {                  cout << "\n\t\t\tAvailable Track: Blackrock Circuit"                       << "\n\t\t\tDefaulting to said track\n";                  counter.blackrock++;                  return "Blackrock Circuit";          }          else if (raceFormat == "Sprint Race") {                  cout << "\n\t\t\tAvailable Track: Rushline Dash"                            << "\n\t\t\tDefaulting to said track\n";                  counter.rushlineDash++;                  return "Rushline Dash";          }          else if (raceFormat == "Drag Race") {                  cout << "\n\t\t\tAvailable Track: Torque Strip"                            << "\n\t\t\tDefaulting to said track\n";                  counter.torqueStrip++;                  return "Torque Strip";          }          return "Track"; // Should never get to this point  }  void setEngineCapacity(int driverCount, string driverName[], int driverAge[], char license[], int engineCapacity[])  {          for (int i = 0; i < driverCount; i++) {                  cout << "\n\t\t\tDriver : " << driverName[i];                  displayEngineCapacities(driverAge[i], license[i]);                  if (driverAge[i] < 13) {                          cout << "\n\t\t\tDriver under 13: Go-Kart must be under 200cc"                                    << "\n\t\t\tDefaulting to 120cc\n";                          engineCapacity[i] = 120;                          counter.cc120++;                          continue;                  }                  // Over 13 without License                  if (license[i] != 'Y') {                          cout << "\n\t\t\tChoose your desired engine capacity (1 - 2): ";                          engineCapacity[i] = getValidIntegerInput("engine capacity", 1 , 2);                  }                  // Over 13 with License                  if (license[i] == 'Y') {                  cout << "\n\t\t\tChoose your desired engine capacity (1 - 3): ";                  engineCapacity[i] = getValidIntegerInput("engine capacity", 1 , 3);                  }                  switch (engineCapacity[i]) {                          case 1: engineCapacity[i] = 120; counter.cc120++; break;                          case 2: engineCapacity[i] = 200; counter.cc200++; break;                          default: engineCapacity[i] = 270; counter.cc270++; break;                  }          }  }  int setLaps(int driverCount, string& raceFormat) {          int laps;          if (raceFormat == "Circuit Race") {                  cout << "\n\t\t\tAvailable number of laps: 2, 3, 4"                            << "\n\t\t\tHow many laps would you like? (2/3/4): ";                  cin >> laps;          }          else if (raceFormat == "Sprint Race" || raceFormat == "Drag Race") {                  cout << "\n\t\t\tSprint Races and Drag Races only have 1 lap\n";                  laps = 1;          }          else if (raceFormat == "Time Trial") {                  cout << "\n\t\t\tAvailable number of laps: Unlimited"                            << "\n\t\t\tHow many laps would you like?: ";                  cin >> laps;          }          else { // Elimination Race                  cout << "\n\t\t\tNumber of laps corresponds with the number of drivers";                  laps = driverCount - 1;                  cout << "\n\t\t\tLaps = Driver Count - 1"                            << "\n\t\t\tLaps = " << driverCount << " - 1"                            << "\n\t\t\tLaps = " << laps << "\n";          }          return laps;  }  string getValidSizeInput(string inputType)  {          string sizeInput;          cin >> sizeInput;          std::transform(sizeInput.begin(), sizeInput.end(), sizeInput.begin(), ::toupper);          while (cin.fail() || (sizeInput != "S" && sizeInput != "M" && sizeInput != "L" && sizeInput != "XL")) {                  cin.clear();                  cin.ignore(1000, '\n');                  cout << "\n\t\t\tPlease choose a valid " << inputType << " (S/M/L/XL): ";                  cin >> sizeInput;                  std::transform(sizeInput.begin(), sizeInput.end(), sizeInput.begin(), ::toupper);          }            return sizeInput;  }  float selectHelmet(int i, string helmetSize[], float helmetPrice[])  {          cout << "\n\t\t\tPlease choose your helmet size (S/M/L/XL): ";          helmetSize[i] = getValidSizeInput("helmet size");          if (helmetSize[i] == "S")                  helmetPrice[i] = 2;          if (helmetSize[i] == "M")                  helmetPrice[i] = 4;          if (helmetSize[i] == "L")                  helmetPrice[i] = 6;          if (helmetSize[i] == "XL")                  helmetPrice[i] = 8;          return helmetPrice[i];  }  float selectSuit(int i, string suitSize[], float suitPrice[])  {          cout << "\n\t\t\tPlease choose your suit size (S/M/L/XL): ";          suitSize[i] = getValidSizeInput("suit size");          std::transform(suitSize[i].begin(), suitSize[i].end(), suitSize[i].begin(), ::toupper);          while (cin.fail() || (suitSize[i] != "S" && suitSize[i] != "M" && suitSize[i] != "L" && suitSize[i] != "XL")){                  cin.clear();                  cin.ignore(1000, '\n');                  cout << "\n\t\t\tPlease choose a valid suit size (S/M/L/XL): ";                  cin >> suitSize[i];                  std::transform(suitSize[i].begin(), suitSize[i].end(), suitSize[i].begin(), ::toupper);          }            if (suitSize[i] == "S")                  suitPrice[i] = 5;          if (suitSize[i] == "M")                  suitPrice[i] = 10;          if (suitSize[i] == "L")                  suitPrice[i] = 15;          if (suitSize[i] == "XL")                  suitPrice[i] = 20;            return suitPrice[i];  }  float selectShoe(int i, int shoeSize[], float shoePrice[])  {          cout << "\n\t\t\tPlease choose your shoe size (20 - 35): ";          cin >> shoeSize[i];            while (cin.fail() || (shoeSize[i] < 20 || shoeSize[i] > 35)){                  cin.clear();                  cin.ignore(1000, '\n');                  cout << "\n\t\t\tPlease choose a valid shoe size (20 - 35): ";                  cin >> shoeSize[i];          }            if (shoeSize[i] <= 25)                  shoePrice[i] = 3;          else                  shoePrice[i] = 6;          return shoePrice[i];  }  float calculateTotalGearPrice(int& driverCount, float helmetPrice[], float suitPrice[], float shoePrice[], float gearPrice[])  {          float totalGearPrice = 0;            for (int i = 0; i < driverCount; i++) {                  gearPrice[i] = helmetPrice[i] + suitPrice[i] + shoePrice[i];                  totalGearPrice += gearPrice[i];          }          return totalGearPrice;  }  float setMembershipDiscount()  {          char membership;          cout << "\n\t\t\tDo you have a membership? (Y/N): ";          membership = getValidCharacterInput();          if(membership == 'Y') {                  counter.membership++;                  return 0.1;          }          else {                  return 0.0;          }  }  float calculatePrice(int engineCapacity[], int laps, int driverCount, float totalGearPrice, float membershipDiscount)  {          float totalKartPrice = 0;            for (int i = 0; i < driverCount; i++)                  totalKartPrice += (engineCapacity[i] \* PRICEPERCC) \* laps;            float finalPrice = (totalKartPrice + totalGearPrice) \* (1 - membershipDiscount);          return finalPrice;  } |

# **6.0 SAMPLES OF INPUT AND OUTPUT**

|  |
| --- |
|  |

# **7.0 DISCUSSIONS**

The Go-Kart Booking System works by interacting with the user to improve the user experiences. We took few cases into our consideration during creating the system. For example, rich or regular customers are familiar with the price for each format or each gear. However, there are first-time customers or custoemrs with low-budget that wanted to try and experience playing Go-Kart. Therefore, we created a display menu that shows race formats, track lists, and price for every racing gear available. This feature will definitely help customers make their bookings.

Other than that, we also added few ASCII arts to make it more attractive and unique. It makes the interface of our booking system became readable even from afar. However, too many usage of the art will make the system looks weird. Therefore, we chose to use the art at specific places such as at the start of the booking system. It acts as the title for our system that will help avoid confusion with other system.

Another problem or issue we faced was the error that occurs when user entered the wrong input. In order to minimise the error, we instruct the user to enter only short answer. For instance, the user only has to input either ‘Y’ or ‘N’ for the license part instead of entering “Yes” or “No”. By doing this, we managed to reduce the error such as wrong spelling of the word. Plus, we also set the choice for race formats by using numbers to avoid misspelling of the format.

# **8.0 CONLUSION**

All in all, Go-Kart Booking System was a huge achievement for us. All sweat and tears are paid off. The system can be run smoothly and successfully without any error and all objectives are achieved. It managed to solve problems faced by the conventional booking system such as human error, time consuming and inefficient. Other than making the booking process more efficient, this sytem also helps the Go-Kart business sustain in this modern world where technology and AI are applied in every aspect. This system can also be applied for other booking system since it has the basic needs to make a booking. It is highly recommended for anyone to improve the system to a whole new dimension in order to create a more complex and advance system.

# **9.0 PRESENTATION**

The presentation is available on YouTube: [INSERT VIDEO LINK]

# **10.0 REFERENCES**

Bro Code. (2024, May 20). *STRUCTS in C++ explained 🏗️* [Video]. YouTube. <https://www.youtube.com/watch?v=ObUUe9ujYoY>

GeeksforGeeks. (2024, March 29). *Why “using namespace std” is considered bad practice*. GeeksforGeeks. <https://www.geeksforgeeks.org/cpp/using-namespace-std-considered-bad-practice/>

Hu, J. (2024, February 23). *How to validate user input in C++*. Delft Stack. <https://www.delftstack.com/howto/cpp/cpp-input-validation/>

Malik, D. S. (2017). *C++ Programming: From problem analysis to program design.* Cengage Learning.

shahxvi. (2025). *csc-126-group-project.* (n.d.). <https://github.com/shahxvi/csc126-group-project>

*std::transform - cppreference.com*. (n.d.). <https://en.cppreference.com/w/cpp/algorithm/transform.html>

Stroustrup, B. (2011). *PPP Style Guide*. <https://www.stroustrup.com/Programming/PPP-style.pdf>

*Text to ASCII: The best ASCII Art Generator & Maker*. (n.d.). Text to ASCII: The Best ASCII Art Generator & Maker. <https://www.asciiart.eu/text-to-ascii-art>

*What’s the problem with “using namespace std;”?* (n.d.). Stack Overflow. <https://stackoverflow.com/questions/1452721/whats-the-problem-with-using-namespace-std>

*W3Schools.com. (n.d.-a).* [*https://www.w3schools.com/cpp/cpp\_enum.asp*](https://www.w3schools.com/cpp/cpp_enum.asp)

*W3Schools.com*. (n.d.-b). <https://www.w3schools.com/cpp/cpp_structs.asp>