```
# include "prjcee.h"

int app_num(struct f1player_se* head) {
	int count = 0;
	struct f1player_se* temp;
	temp = (struct f1player_se*)malloc(sizeof(struct f1player_se));
	temp = head;
	while (temp != NULL) {
		count++;
		temp = temp->next;
	}
	return count;
}

int swap(struct f1player_se* one, struct f1player_se* two, struct f1player_se*
three) {
	if (two->wins < three->wins) {
		two->next = three->next;
		three->next = two;
		one->next = three;
		// one
		// three
		// two
		return 1;
	}
}

struct f1player_se* sort_wins(struct f1player_se* head)
{
	struct f1player_se* temp;
	struct f1player_se* one = head;
	struct f1player_se* two = head->next;
	struct f1player_se* three = two->next;
	int i;
	int count = 1;
	for (i = 0; i < 21; i++) {
		one = head;
		two = head->next;
		three = two->next;
		if ((head->next)->wins > head->wins) {
			temp = head;
			head = head->next;
			head->next = temp;
			temp->next = three;
			two = one->next;
			three = two->next;

			printf("First node was swapped\n");
			printf("one: %d\n", one->wins);
			printf("two: %d\n", two->wins);
			printf("three: %d\n\n", three->wins);
			count++;
		}
		else {
			printf("First node was not swapped\n");
			printf("one: %d\n", one->wins);
			printf("two: %d\n", two->wins);
			printf("three: %d\n", three->wins);
		}
```

```
            while (three->next != NULL) {
                  printf("count: %d\n", count);
                  printf("Before\n");
                  printf("one: %d\n", one->wins);
                  printf("two: %d\n", two->wins);
                  printf("three: %d\n", three->wins);
                  printf("three->next: %d\n", (three->next)->wins);
                  if (swap(one, two, three) == 1) {
                        printf("Operation performed: Swapped\n");
                        one = three;
                        three = two->next;
                  }
                  else {
                        printf("Operation performed: Shifted\n");
                        one = one->next;
                        two = two->next;
                        three = three->next;
                  }
                  printf("After\n");
                  printf("one: %d\n", one->wins);
                  printf("two: %d\n", two->wins);
                  printf("three: %d\n\n", three->wins);
                  count++;
            }
            if (two->wins < three->wins) {
                  two->next = NULL;
                  one->next = three;
                  three->next = two;
            }
            printf("Bottom of list\n\n");
      }
      return head;
}

struct f1player_se* hiring(struct f1player_se* head, char* position)
{
      // Sorts drivers into their desired positions then hires based on who has the
highest wins
      // within each desired position group

      struct f1player_se* ptr;
      struct f1player_se* sub_head;
      struct f1player_se* test_head;

      struct top_app* main_head;
      struct top_app* main_temp;
      struct top_app* temp;
      ptr = head;
      int count = 0;

      while (ptr != NULL) {
            if ((ptr->pos_desired)[0] == position) {
                  count++;
            }
            ptr = ptr->next;
      }

      if (count == 0) {
```

```
            if (position == 'm') {
                  printf("Not enough main driver applicants left\n");
            }
            if (position == 's') {
                  printf("Not enough substitute driver applicants left\n");
            }
            if (position == 't') {
                  printf("Not enough test driver applicants left\n");
            }
            return 0;
      }

      ptr = head;
      while ((ptr->pos_desired)[0] != position) {
            ptr = ptr->next;
      }


      main_head = (struct top_app*)malloc(sizeof(struct top_app));
      main_head->id = ptr->id;
      main_head->wins = ptr->wins;
      main_head->next = NULL;

      main_temp = (struct top_app*)malloc(sizeof(struct top_app));
      main_temp = main_head;

      while (ptr != NULL) {
            if ((ptr->pos_desired)[0] == position) {
                  temp = (struct top_app*)malloc(sizeof(struct top_app));
                  temp->id = ptr->id;
                  temp->wins = ptr->wins;
                  temp->next = main_temp;
                  main_temp = temp;
            }
            ptr = ptr->next;
      }

      // main_temp is at the beginning

      int highest_wins = main_temp->wins;
      int highest_id = main_temp->id;
      main_temp = main_temp->next;

      while (main_temp != NULL) {
            if (main_temp->wins > highest_wins) {
                  highest_wins = main_temp->wins;
                  highest_id = main_temp->id;
            }
            main_temp = main_temp->next;
      }

      return highest_id; // The hired driver

}

struct f1player_se* process(struct f1player_se* head, int clock_time)
{
      int count = 0;
      int i;
```

```
	struct f1player_se* ptr;
	struct f1player_se* ptr_next;
	struct f1player_se* ptr_current;
	struct f1player_se* ptr_prev;

	ptr = head;

	while (ptr != NULL) {
		if (ptr->time_in > 0) {
			head = del_app(head, ptr->id);
		}
		ptr = ptr->next;
	}

	int main_id = hiring(head, 'm');
	int sub_id = hiring(head, 's');
	int test_id = hiring(head, 't');

	if (main_id != 0) {
		printf("Applicant %d was hired for Main driver\n", main_id);
	}
	if (sub_id != 0) {
		printf("Applicant %d was hired for Sub driver\n", sub_id);
	}
	if (test_id != 0) {
		printf("Applicant %d was hired for Test driver\n", test_id);
	}

	// Counts current number of applicants and increments timein for everyone
	ptr = head;
	while (ptr != NULL) {
		count++;
		ptr->time_in++;
		ptr = ptr->next;
	}
	// Inserts new applicants at the end
	ptr = head;
	for (i = 1; i < count; i++) {
		ptr = ptr->next;
	}
	ptr->next = new_apps(clock_time);

	head = del_app(head, main_id);
	head = del_app(head, sub_id);
	head = del_app(head, test_id);

	ptr = head;
	while (ptr != NULL) {
		if (ptr->wins < 11) {
			head = del_app(head, ptr->id);
		}
		ptr = ptr->next;
	}

	return head;
}

void print_samerp(struct f1player_se* head, int score)
{
```

```
	struct f1player_se* ptr;
	ptr = head;
	int count = 0;

	printf("Applicants with desired race point:\n");
	while (ptr != NULL) {
		if (ptr->race_point == score) {
			printf("%d\n", ptr->id);
			count++;
		}
		ptr = ptr->next;
	}
	if (count == 0) {
		printf("No applicants have the desired race point\n");
	}
}

del_app (struct f1player_se* head, int id_input)
{
	struct f1player_se* ptr;
	struct f1player_se* ptr_next;
	struct f1player_se* ptr_current;
	struct f1player_se* ptr_prev;

	ptr = head;
	if (ptr->id == id_input) {
		head = ptr->next;
		return head;
	}

	ptr_prev = head;
	while (ptr_prev->next != NULL) {
		ptr_current = ptr_prev->next;
		if (ptr_current->id == id_input) {
			ptr_prev->next = ptr_current->next;
			return head;
		}
		ptr_prev = ptr_prev->next;
	}

	if (ptr_current->id != NULL) {
		if (ptr_current->id == id_input) {
			ptr_prev->next = NULL;
			free(ptr_current);
			return head;
		}
	}
	return head;
}

void analyze_app_list(struct f1player_se* head)
{
	struct f1player_se* ptr = head;
	printf("%3s %6s %s\n", "id", "wins", "probability of getting in");
	while (ptr != NULL) { // Make these a for loop
		if (ptr->wins < 21 && ptr->wins > 10) {
			printf("%d %3d %10s\n", ptr->id, ptr->wins, "Low");
		}
		if (ptr->wins < 31 && ptr->wins > 20) {
```

```
            printf("%d %3d %13s\n", ptr->id, ptr->wins, "Moderate");
      }
      if (ptr->wins < 41 && ptr->wins > 30) {
            printf("%d %3d %16s\n", ptr->id, ptr->wins, "Above Average");
      }
      if (ptr->wins < 51 && ptr->wins > 40) {
            printf("%d %3d %11s\n", ptr->id, ptr->wins, "High");
      }
      ptr = ptr->next;
}
ptr = head;
printf("\n");
// Make this a function
printf("Team Redbull:\n");
while (ptr != NULL) {
      if ((ptr->prev_team)[0] == 'R') {
            printf("%d\n", ptr->id);
      }
      ptr = ptr->next;
}
ptr = head;
printf("Team Alpine:\n");
while (ptr != NULL) {
      if ((ptr->prev_team)[0] == 'A') {
            printf("%d\n", ptr->id);
      }
      ptr = ptr->next;
}
ptr = head;
printf("Team Haas:\n");
while (ptr != NULL) {
      if ((ptr->prev_team)[0] == 'H') {
            printf("%d\n", ptr->id);
      }
      ptr = ptr->next;
}
ptr = head;
printf("Team Porsche:\n");
while (ptr != NULL) {
      if ((ptr->prev_team)[0] == 'P') {
            printf("%d\n", ptr->id);
      }
      ptr = ptr->next;
}
ptr = head;
printf("Team Mercedes:\n");
while (ptr != NULL) {
      if ((ptr->prev_team)[0] == 'M' && (ptr->prev_team)[1] == 'e') {
            printf("%d\n", ptr->id);
      }
      ptr = ptr->next;
}
ptr = head;
printf("Team Mclaren:\n");
while (ptr != NULL) {
      if ((ptr->prev_team)[0] == 'M' && (ptr->prev_team)[1] == 'c') {
            printf("%d\n", ptr->id);
      }
      ptr = ptr->next;
```

```
	}
	ptr = head;

	printf("Group 1:\n");
	while (ptr != NULL) {
		if (ptr->race_point > 749 && ptr->race_point < 1001) {
			printf("%d\n", ptr->id);
		}
		ptr = ptr->next;
	}
	ptr = head;
	printf("Group 2:\n");
	while (ptr != NULL) {
		if (ptr->race_point > 449 && ptr->race_point < 750) {
			printf("%d\n", ptr->id);
		}
		ptr = ptr->next;
	}
	ptr = head;
	printf("Group 3:\n");
	while (ptr != NULL) {
		if (ptr->race_point < 500) {
			printf("%d\n", ptr->id);
		}
		ptr = ptr->next;
	}
}


void terminate_write(struct f1player_se* head)
{
	char buffer[32] = { 0 };
	struct f1player_se* temp;
	temp = head;
	FILE* fp;
	fp = fopen("ter_write.txt", "w");
	while (temp != NULL) {
		sprintf(buffer, "%d ", temp);
		fputs(buffer, fp);
		sprintf(buffer, "%d ", temp->id);
		fputs(buffer, fp);
		sprintf(buffer, "%d ", temp->app_date);
		fputs(buffer, fp);
		sprintf(buffer, "%d ", temp->time_in);
		fputs(buffer, fp);
		sprintf(buffer, "%s ", temp->prev_team);
		fputs(buffer, fp);
		sprintf(buffer, "%s ", temp->pos_desired);
		fputs(buffer, fp);
		sprintf(buffer, "%d ", temp->wins);
		fputs(buffer, fp);
		sprintf(buffer, "%d ", temp->qual_point);
		fputs(buffer, fp);
		sprintf(buffer, "%d ", temp->race_point);
		fputs(buffer, fp);
		sprintf(buffer, "%d ", temp->avg_skill_score);
		fputs(buffer, fp);
		sprintf(buffer, "%d\n", temp->next);
		fputs(buffer, fp);
```

```
            temp = temp->next;
      }
      fclose(fp);
      while (head != NULL)
      {
            temp = head;
            head = head->next;
            free(temp);
      }
}

// I modified print apps to be my print formatted
```