1. Introduction

A. 프로그램 명 : 모멘텀 순위를 기준으로 한 주식 매입 시뮬레이션

B. 개요(요약 및 소개)

1) 요약

이 프로그램은 csv 파일에 담겨 있는 여러 주식의 가격을 기준으로 모멘텀을 계산한다. 계산한 모멘텀을 기준으로 순위를 메기고 사용자가 입력한 순위까지와 매입 날짜에 따라 csv 파일에 있는 제일 최근 날짜의 수익률을 csv 파일에 정한다. csv 파일에 저장된 날 중 최고 수익률이 난 날짜를 계산해 결과를 console에 출력한다. 사용자가 최저 수익률을 입력하면 파일의 제일 최근 날짜 정보 기준 수익률을 계산해 최저 수익률을 만족하는 주식의 매입 시기 모멘텀 값을 출력한다. 또한 사용자가 주식 코드를 입력하면 이 주식에 대한 정보를 console 창에 출력한다.

2) 기대 효과

이 프로그램을 통해 모멘텀을 이용한 주식 매입이 현재 주식가 기준 어느 정도의 수익률을 내는지, 최고 수익률은 언제인지, 그리고 최저 모멘텀 값이 어느 정도 되어야 사용자가 정한 수익률이 나오는지 분석한다.

3) 주식 모멘텀의 정의[1]

금일 가격과 전 가격의 비교하여 주가 추세 속도가 증가하는지 감소하는지 판단하는 지표로 대표적으로 사용되는 평균 모멘텀은 계산법은 다음과 같다. 매달 정해진 시기의 주식 값을 1년 전 주식 값과의 차이의 평균을 계산해 12개월 모멘텀을 계산한다. 12개월 모멘텀을 계산하는 이유는 다른 3개월, 6개월 등 다른 모멘텀 계산에 비해 경험적으로 수익률이 높은 것으로 학계에서 판단하였기 때문이다.

C. 주요 기능 및 상세 기능

1) 입력 파일

주식 데이터 값은 주로 excel 형식 혹은 csv 형식으로 구성되어 있다. 이에 맞춰 입력 파일은 csv 형식으로 통일한다. 만약 excel 파일을 가지고 있다면 이를 csv로 변환 후 사용한다.

2) 12개월 모멘텀 계산 및 정보 가공

파일의 첫날부터 1개월 기준으로 전년도와 비교하여 평균 모멘텀을 계산한다.

계산 식 : (계산 할 시기의 주식가 - 계산 할 시기 기준 1년 전의 주식가)/12

계산된 12개월 평균 모멘텀의 값이 큰 순서대로 순위를 메긴다.

파일의 제일 최근 정보 기준으로 12개월 모멘텀을 계산한다.

3) 사용자 입력

사용자는 모멘텀이 높은 순으로 나열된 주식을 몇 순위까지 매입할 것인지를 입력한다. 그리고 매입할 날짜를 입력한다. 프로그램은 이 주식들을 매입한 것으로 판단하게 되고 매입 날짜 이후의 정보에 대해 수익률을 계산한다. 또한 사용자는 최저 수익률을 입력할 수 있는데 매입 날짜 기준 이 수익률을 이루기 위해서 어느 정도의 모멘텀 값을 갖는 주식을 매입해야 할지 모멘텀 값을 제시한다. 사용자가 만약 주식 코드를 입력하면 이 주식에 한해서만 정보를 console창에 출력한다.

4) 분석 결과 보여주기

매입한 주식의 파일 내 제일 최근 날짜 수익률을 계산해 보여주고 최고 수익은 어느 시기에 발생하였는지 보여준다. 입력 받은 최저 수익률을 내는 주식의 매입 시기 모멘텀 값을 계산해 알려준다. 또 모든 주식에

대한 모멘텀 계산값과 수익률 계산 값이 추가된 csv 파일을 생성한다.

D. 사용자 메뉴

1) 정해진 형식대로 작성된 csv 파일을 입력 파일로 제공한다.

2) 프로그램이 할 일을 입력한다.

(ex. 1. 모든 주식에 대한 정보 출력 2. 주식 매입 및 최저 수익률에 대한 모멘텀 찾기 3. 정해진 주식에 대한 정보 출력)

3) 2)에서 '1' 입력 시 입력 받은 파일의 정보를 가공하지 않고 그대로 출력한다.

4) 2)에서 '2' 입력 시 주식을 매입할 날짜, 순위를 입력하고 이에 대한 현재 수익률을 보고 받는다. 또한 입력 받은 최저 수익률을 갖는 주식에 대해 사용자가 매입한 날 모멘텀을 출력한다.

5) 2-4를 반복하되 만약 F를 입력하면 파일의 제일 최근 시점 12개월 모멘텀을 포함한 csv 파일을 생선하고 프로그램을 종료한다.

2. 프로그램 사용법

a) 함께 첨부된 'all_month_data.csv'를 프로그램과 같은 폴더에 저장한다.(상대경로)

b) 프로그램을 실행시킨다.

c) 주식 구입을 원하는 날짜를 입력한다.

데이터가 2015년 5월 14일부터 2019년 5월 14일까지만 저장되어 있기 때문에 이 사이 값을 입력해야 하며 만약 다른 값을 입력하면 다시 입력을 받는다. 또한 입력 형태는 20190514와 같이 오롯이 숫자로 이루어진 8자리 숫자여야 한다. 2016년 5월 14일 이전 날짜를 입력하면 12개월 전 데이터를 분석할 수 없기 때문에 데이터가 정확하지 않다는 에러 메시지를 주고 계산을 진행한다. 날짜는 14일만 입력 받고 14일이 아닌 날짜를 입력하면 다시 입력 받는다.

d) 입력한 날짜 기준 평균 모멘텀을 기준으로 기록된 순위 중 어느 순위까지 주식을 구입할지 입력한다.

e) 원하는 최저 수익률을 입력한다. 수익률은 소수점의 형태로 입력해야 한다.

f) 순위 내 주식에 대해 주식 순위, 주식 번호, 수익률과 입력한 날짜에 구입했다 가정했을 때 파일의 제일 최근 날짜인 2019년 5월 14일의 수익률, 최고 수익률이 발생한 날과 최고 수익률을 보고받는다.

g) 모든 주식에 대해 구매한 날 기준으로 수익률이 최저 수익률보다 높으면 그 주식 번호와 모멘텀을 보고 받는다. 없다면 표시되지 않는다.

h) 어떤 일을 할지 모드를 선택한다

1. 파일에 저장된 데이터를 console에 출력 받는다.

2. c)~g)를 다시 반복한다.

3. 원하는 주식의 번호를 입력하고 가장 최근에 입력한 구입 날짜에 의한 평균 모멘텀, 순위, 날짜와 그 날짜의 가격을 console에 출력 받는다.

F. 가장 최근 입력한 구입 날짜에 의한 데이터를 'final_data.csv'파일에 저장하고 프로그램을 종료한다.

i) 'F'를 입력하기 전까지 mode 선택을 계속한다.

3. Optimization

profiling 결과를 측정할 때는 콘솔창 input은 20170514, 6, 0.2로 하고 'finished finding stocks'가 출력된 후 input은 1로 한다. row data를 모두 출력한 후 F를 입력하여 프로그램을 종료한다.

compiler used: gcc (Ubuntu 7.5.0-3ubuntu1~18.04) 7.5.0

option: -Og -pg -o (output 파일 이름: op(번호))

encoding: UTF-8

입력: 입력 파일-주식 번호, 날짜, 가격이 순서대로 저장되어 있는 csv 파일

console 입력-위에 명시된 대로 20170514, 6, 0.2, 1, F

출력: console 출력-구한 주식의 profit, 정보 등

출력 파일(final_data.csv)-주식 번호, 평균 모멘텀, 랭킹/날짜, 가격, 모멘텀, 이익

```
line : 490
stock variety : 10
enter the day you want to buy stocks(20150514 ~ 20190514)
20170514
enter the rank you want to buy.
6
enter minimum profit you want
0.2

rank 1 stock no. 5930
date : profit
```

그림 1 profiling 결과 측정을 위한 input 1

```
finished finding stocks

what to do?
1. print row data
2.buy stock and find momentum that gives minimum profit
3.print data of specific stock
 F: terminate1
stock   date      price

660     20190514        74600
660     20190414        79700
660     20190314        67300
```

그림 2 profiling 결과 측정을 위한 input 2

```
what to do?
1. print row data
2.buy stock, find momentum that gives minimum profit
3.print data of specific stock
 F: terminateF

program terminated
```

그림 3 profiling 결과 측정을 위한 input 3

A. no optimization (row code)

1) source code

```
#include <stdio.h>
#include <stdlib.h>
typedef struct a_stock {
    int stock_no;   //   stock's number which is used to distinct stocks
    int date[50];   //   array to save date of the data
    int price[50];   //   array to save price of the data
    int momentum[50];   //   array to save calculated momentum
    double profit[50];   //   array to save calculated profit
    int avg_momentum;   //   to save the 12 month average momentum of momentum
    int rank;   //   to save the ranking of the stock ranked by average momentum
}STOCK;

typedef struct ROW_DATA {
    int a_stock_no;   //   stock's number from the file
    int a_date;   //   date of recoding
    int a_price;   //   price of the stock of the day
} ROW_DATA;

// function prototype
int count_line(FILE *infile);
int count_variety(FILE *infile, ROW_DATA *row_data, int line);
void data_selection(FILE *infile, ROW_DATA *row_data, STOCK *select_data, int line);
void calculate_momentum(FILE *infile, ROW_DATA *row_data, STOCK *select_data, int
buy_date, int line);
void calculate_profit(FILE *infile, ROW_DATA *row_data, STOCK *select_data, int buy_rank,
double min_profit, int buy_date, char work, int line);
void print_row_data(FILE *infile, ROW_DATA *row_data);
void print_the_data(FILE *infile, ROW_DATA *row_data, STOCK *select_data, int print_stock,
int line);
void make_output_file(FILE *infile, ROW_DATA *row_data, FILE *outfile, STOCK *select_data,
int line);
void get_user_input(FILE *infile, int *buy_date, int *buy_rank, double *min_profit, int
*print_stock, ROW_DATA *row_data, STOCK *select_data, FILE *outfile, int line);


int main(void) {
```

```
	//	pointer to handle files
	FILE *infile = fopen("all_month_data.csv", "r");	//	input file
	FILE *outfile = fopen("final_data.csv", "w");	//	output file

	//	check file opening succeed
	if(infile == NULL || outfile == NULL){
		printf("file opening failed₩n");
		return 1;
	}

	int line = count_line(infile);	//	whole line of file, same as data's no.
	printf("line : %d₩n", line);



	//	pointer array to save data from infile
	ROW_DATA *row_data = (ROW_DATA *)malloc(sizeof(ROW_DATA) * line);
	printf("stock variety : %d₩n", count_variety(infile, row_data, line));
//	pointer array to save collected data
	STOCK *select_data = (STOCK *)malloc(sizeof(STOCK)*count_variety(infile, row_data,
line));

	//	check if dynamic memory allocation is succeed
	if(row_data == NULL || select_data == NULL){
		printf("allocating dynamic memory failed₩n");
		return 2;
	}

	// variables to save user's input
	int buy_date, buy_rank, print_stock;
	double min_profit;

	//	assign data of row_data to select data to combine data
	data_selection(infile, row_data, select_data, line);

	//	get keyboard input from user
	get_user_input(infile, &buy_date, &buy_rank, &min_profit, &print_stock, row_data,
				select_data, outfile, line);

	//	free dynamic data of pointer arrays
```

```
	free(select_data);
	free(row_data);

	//	close file
	fclose(infile);
	fclose(outfile);

	//	terminate the program
	return 0;
}

void get_user_input(FILE *infile, int *buy_date, int *buy_rank, double *min_profit,
				int *print_stock, ROW_DATA *row_data, STOCK *select_data, FILE
*outfile, int line) {
	/*
	 * variable to select mode
	 * to get rank and average momentum, user should input purchasing date
	 * so program runs mode 2 first
	 */
	char work = '2';

	while (1) {
		if (work == '1') {
			//	print the data of row_data
			print_row_data(infile, row_data);

			//	repeat selecting mode
			printf("what to do?₩n");
			printf("1. print row data ₩n");
			printf("2.buy stock, find momentum that gives minimum profit₩n");
			printf("3.print data of specific stock₩n F: terminate");
			scanf(" %c", &work);
		} else if (work == '2') {
			printf("enter the day you want to buy stocks(20150514 ~ 20190514)₩n");
			scanf("%d", buy_date);

			/*
			 * if buy_date is not 14th, get it again
			 * if buy_date is not in file, get it again
```

```
                 */
                while ((((*buy_date - 14) % 100) != 0) || (*buy_date < 20150513)
                        || (*buy_date > 20190515)) {
                    printf("improper input input date again₩n");
                    scanf("%d", buy_date);
                }

                //  give warning message if user ask data for last 12 months
                if (*buy_date < 20160514) {
                    printf("average momentum, rank is not effective ₩n");
                    printf("because there is no data to calculate momentum₩n");
                }

                printf("enter the rank you want to buy.₩n");
                scanf("%d", buy_rank);
                printf("enter minimum profit you want₩n");
                scanf("%lf", min_profit);

                //  calculate momentum and average momentum
                calculate_momentum(infile, row_data, select_data, *buy_date, line);

                //  calculate profit and print max profit and the date of max
                calculate_profit(infile, row_data, select_data, *buy_rank, *min_profit,
*buy_date, work, line);

                //  repeat selecting mode
                printf("what to do?₩n");
                printf("1. print row data ₩n");
                printf("2.buy stock and find momentum that gives minimum profit₩n");
                printf("3.print data of specific stock₩n F: terminate");
                scanf(" %c", &work);
            } else if (work == '3') {
                //  get stock no. to print
                printf("enter number of stocks that you want to have the data of₩n");
                scanf("%d", print_stock);

                //  print the data
                print_the_data(infile, row_data, select_data, *print_stock, line);
```

```
            //   repeat selecting mode
            printf("what to do?₩n");
            printf("1. print row data ₩n");
            printf("2.buy stock and find momentum that gives minimum profit₩n");
            printf("3.print data of specific stock₩n F: terminate");
            scanf(" %c", &work);
        } else if (work == 'F') {
            //   calculate momentum of all stocks
            calculate_momentum(infile, row_data, select_data, *buy_date, line);
            //   calculate profit of all stocks and print the data to outfile
            make_output_file(infile, row_data, outfile, select_data, line);
            printf("₩nprogram terminated₩n");
            break;
        } else {
            //   get mode input again
            printf("improper input₩n₩n");
            printf("what to do?₩n");
            printf("1. print row data ₩n");
            printf("2.buy stock and find momentum that gives minimum profit₩n");
            printf("3.print data of specific stock₩n F: terminate");
            scanf(" %c", &work);
        }
    }
}
int count_line(FILE *infile) {
    int line = 0;
    int ch;   //   to save char gotten by fgetc()

    while (!feof(infile)) {
        ch = fgetc(infile);
        if (ch == '₩n') {
            line++;
        }
    }

    //printf("line : %d₩n", line);   //   print line to check the value
    rewind(infile);   //   rewind infile to get data from it

    return line;   //   return line
```

```
}
int count_variety(FILE *infile, ROW_DATA *row_data, int line) {
	int a_day, a_price, a_stock_no;   //   variables to get data from file
	int variety = 0;   // initialized as 0 since calculation contains plus

	//printf("line %d", count_line(infile));
	for (int i = 0; i < line; i++) {
		fscanf(infile, "%d,%d,%d₩n", &a_stock_no, &a_day, &a_price);

		row_data[i].a_date = a_day;
		row_data[i].a_stock_no = a_stock_no;
		row_data[i].a_price = a_price;
	}

	rewind(infile);
	for (int i = 0; i < count_line(infile); i++) {
		if ((row_data[i].a_stock_no) != (row_data[i + 1].a_stock_no)) {
			variety++;
		}
	}

	rewind(infile);
	return variety;   //   return variety
}
void data_selection(FILE *infile, ROW_DATA *row_data, STOCK *select_data, int line) {
	int stock_variety = count_variety(infile, row_data, line);
	for (int i = 0; i < stock_variety; i++) {
		int month = line/stock_variety;
		for (int j = 0; j < month*stock_variety; j++) {
			select_data[i].stock_no = row_data[month*i].a_stock_no;

			for (int k = 0; k < month; k++) {
				select_data[i].date[k] = row_data[month*i + k].a_date;
				select_data[i].price[k] = row_data[month*i + k].a_price;
				select_data[i].rank = 1;   //   initialize rank as 1
				select_data[i].momentum[k] = 0;   //   initialize momentum as 0
				select_data[i].profit[k] = 0;   //   initialize profit as 0
			}
		}
```

```
    }
}

void calculate_momentum(FILE *infile, ROW_DATA *row_data, STOCK *select_data, int
buy_date, int line) {
    int total = 0;   //   initialize total to calculate average
    int stock_variety = count_variety(infile, row_data, line);
    for (int i = 0; i < stock_variety; i++) {
        int month = line/stock_variety;
        for (int j = 0; j < month -12; j++) {
            select_data[i].momentum[j] = select_data[i].price[j] -
                                                    select_data[i].price[j + 12];
        }
    }
for (int i = 0; i < stock_variety; i++) {
        int month = line/stock_variety;
        for (int j = 0; j < month; j++) {
            if (select_data[i].date[j] == buy_date) {
                for (int k = 0; k < 12; k++) {
                    total += select_data[i].momentum[j + k];
                }
            }
        }
        select_data[i].avg_momentum = (double)total/12;
        total = 0;
    }

for (int i = 0; i < stock_variety; i++) {
        for (int j = 0; j < stock_variety; j++) {
            if (select_data[i].avg_momentum < select_data[j].avg_momentum) {
                select_data[i].rank++;
            }
        }
    }
}

void calculate_profit(FILE *infile, ROW_DATA *row_data, STOCK *select_data, int buy_rank,
double min_profit, int buy_date, char work, int line) {
    //   to get profit for the stock in rank
```

```
for (int k = 1; k <= buy_rank; k++) {
    for (int i = 0; i < count_variety(infile, row_data, line); i++) {
        //  variables to get max profit
        double max_profit = select_data[i].profit[0];
        //  variable to get max profit's index
        int max_num;
        int month = line/count_variety(infile, row_data, line);

        //  calculate profit for stock whose rank is 1 to buy_rank
        if (select_data[i].rank == k) {
            printf("₩nrank %d stock no. %d₩n", k, select_data[i].stock_no);
            printf("date : profit₩n₩n");

            for (int j = 0; j < month; j++) {
                select_data[i].profit[j] = (select_data[i].price[0] -
                                            select_data[i].price[j])/
                                            (double)select_data[i].price[j];
                //  print the profit
                printf("%d : %lf₩n", select_data[i].date[j],
                            select_data[i].profit[j]);

            }

            //  find max profit
            for (int j = 0; j < month; j++) {
                    if (max_profit < select_data[i].profit[j]) {
                        max_profit = select_data[i].profit[j];
                        max_num = j;
                    }

                //  print rank, stock_no, profit of purchasing date
                if (select_data[i].date[j] == buy_date) {
                    printf("₩nbuy_date's profit: %lf₩n",
                            select_data[i].profit[j]);
                }
            }
            // print the maximum profit
            printf("₩nrank%d max profit: %lf, date : %d₩n₩n", k,
                        max_profit, select_data[i].date[max_num]);
```

```
                }
            }
        }
        printf("momentum of purchasing date that satisfies minimum₩n");
        for (int i = 0; i < count_variety(infile, row_data, line); i++) {
            int month = line/count_variety(infile, row_data, line);
            for (int j = 0; j < month; j++) {
                if (select_data[i].date[j] == buy_date) {
                    if (min_profit <= select_data[i].profit[j]) {
                        printf("profit %d : %d₩n", select_data[i].stock_no,
                            select_data[i].momentum[j]);
                    }
                }
            }
        }

        //  mark finishing find stocks that satisfy minimum profit
        printf("₩nfinished finding stocks₩n₩n");
}

void print_row_data(FILE *infile, ROW_DATA *row_data) {
        printf("stock₩tdate₩tprice₩n₩n");
        int line = count_line(infile);
        int month = line/count_variety(infile, row_data, line);
        for (int i = 0; i < line; i++) {
            printf("%d₩t%d₩t%d₩n", row_data[i].a_stock_no, row_data[i].a_date,
                    row_data[i].a_price);
        }
}

void print_the_data(FILE *infile, ROW_DATA *row_data, STOCK *select_data, int print_stock,
int line) {
        for (int i = 0; i < count_variety(infile, row_data, line); i++) {
            if (select_data[i].stock_no == print_stock) {
                //  print title(average momentum, ranking of stock)
                printf("average momentum : %d, rank : %d₩n",
                        select_data[i].avg_momentum, select_data[i].rank);
                //  print name of the data
```

```
                printf("date₩tprice₩n");
                int month = line/count_variety(infile, row_data, line);
                for (int j = 0; j < month; j++) {
                    //  print date and price of the data
                    printf("%d₩t%d₩n", select_data[i].date[j],
                            select_data[i].price[j]);
                }
            }
        }
}

void make_output_file(FILE *infile, ROW_DATA *row_data, FILE *outfile, STOCK *select_data,
int line) {
    //  calculate profit for all stocks
    for (int i = 0; i < count_variety(infile, row_data, line); i++) {
        int month = line/count_variety(infile, row_data, line);
        for (int j = 0; j < month; j++) {
            select_data[i].profit[j] = (select_data[i].price[0] -
                                        select_data[i].price[j])/
                                        (double)select_data[i].price[0];
        }
    }

    //  loop for a type of stock
    for (int i = 0; i < count_variety(infile, row_data, line); i++) {
        int month = line/count_variety(infile, row_data, line);
        //  print general information of the stock
        fprintf(outfile, "%d%c%d%c%d₩n", select_data[i].stock_no, ',',
                    select_data[i].avg_momentum, ',', select_data[i].rank);

        //  print specific data of the stock
        for (int j = 0; j < month; j++) {
            fprintf(outfile, "%d%c%d%c%d%c%lf₩n", select_data[i].date[j], ',',
select_data[i].price[j], ',', select_data[i].momentum[j],
                    ',', select_data[i].profit[j]);
        }
    }
}
```

2) profiling 결과

```
dw@dw-VirtualBox:/media/sf_2020/SystemProgram/HW3/code_optimization$ gprof row_code1
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 92.94      0.13     0.13    95747     0.00     0.00  count_line
  7.15      0.14     0.01        1    10.01    10.68  data_selection
  0.00      0.14     0.00      195     0.00     0.67  count_variety
  0.00      0.14     0.00        2     0.00     0.67  calculate_momentum
  0.00      0.14     0.00        1     0.00    98.09  calculate_profit
  0.00      0.14     0.00        1     0.00   128.12  get_user_input
  0.00      0.14     0.00        1     0.00    28.03  make_output_file
  0.00      0.14     0.00        1     0.00     0.67  print_row_data
```

그림 4 no optimization profile result

```
granularity: each sample hit covers 2 byte(s) for 7.14% of 0.14 seconds

index % time    self  children    called     name
                                                 <spontaneous>
[1]    100.0    0.00    0.14                 main [1]
                0.00    0.13       1/1           get_user_input [4]
                0.01    0.00       1/1           data_selection [7]
                0.00    0.00       2/195         count_variety [3]
                0.00    0.00       1/95747       count_line [2]
-----------------------------------------------
                0.00    0.00       1/95747       print_row_data [9]
                0.00    0.00       1/95747       main [1]
                0.13    0.00   95745/95747       count_variety [3]
[2]     92.9    0.13    0.00   95747         count_line [2]
-----------------------------------------------
                0.00    0.00       1/195         data_selection [7]
                0.00    0.00       1/195         print_row_data [9]
                0.00    0.00       2/195         calculate_momentum [8]
                0.00    0.00       2/195         main [1]
                0.00    0.03      42/195         make_output_file [6]
                0.00    0.10     147/195         calculate_profit [5]
[3]     92.9    0.00    0.13     195         count_variety [3]
                0.13    0.00   95745/95747       count_line [2]
-----------------------------------------------
                0.00    0.13       1/1           main [1]
[4]     91.4    0.00    0.13       1         get_user_input [4]
                0.00    0.10       1/1           calculate_profit [5]
                0.00    0.03       1/1           make_output_file [6]
                0.00    0.00       2/2           calculate_momentum [8]
                0.00    0.00       1/1           print_row_data [9]
-----------------------------------------------
                0.00    0.10       1/1           get_user_input [4]
[5]     70.0    0.00    0.10       1         calculate_profit [5]
                0.00    0.10     147/195         count_variety [3]
-----------------------------------------------
                0.00    0.03       1/1           get_user_input [4]
[6]     20.0    0.00    0.03       1         make_output_file [6]
                0.00    0.03      42/195         count_variety [3]
-----------------------------------------------
                0.01    0.00       1/1           main [1]
[7]      7.6    0.01    0.00       1         data_selection [7]
                0.00    0.00       1/195         count_variety [3]
-----------------------------------------------
-----------------------------------------------
                0.00    0.00       2/2           get_user_input [4]
[8]      1.0    0.00    0.00       2         calculate_momentum [8]
                0.00    0.00       2/195         count_variety [3]
-----------------------------------------------
                0.00    0.00       1/1           get_user_input [4]
[9]      0.5    0.00    0.00       1         print_row_data [9]
                0.00    0.00       1/195         count_variety [3]
                0.00    0.00       1/95747       count_line [2]
-----------------------------------------------
```

그림 5 no optimization call graph

3) profiling 결과 분석

그림 1을 보면 다른 함수의 call 횟수는 납득할 수 있는 것에 반해 count_line과 count_variety의 call 횟수가 비정상적으로 많은 것을 알 수 있고 count_line이 run time의 92.94%를 차지하고 있다. 그림 2을 봤을 때 count_line을 호출한 건 count_variety가 압도적으로 많다.

4) 다음 단계 optimization 내용

위 분석을 토대로 할 때 count_variety 내 count_line의 호출 횟수를 줄이는 것이 다음 optimization 목표이다.

B. optimization 1

1) source code

```
327     /*
328      * since same stock's data is saved in series,
329      * count the time that stock's number and next array's stock's number is
330      * different
331      */
332     rewind(infile);
333     for (int i = 0; i < count_line(infile); i++) {
334         if ((row_data[i].a_stock_no) != (row_data[i + 1].a_stock_no)) {
335                 variety++;
336             }
337     }
338
339     //printf("stock variety : %d\n", variety);  //  check the variety
340     rewind(infile);
341     return variety;  //  return variety
342 }
```

그림 6 count_variety op1 바꿀 부분

```
    for (int i = 0; i < line; i++) {
        if ((row_data[i].a_stock_no) != (row_data[i + 1].a_stock_no)) {
                variety++;
            }
    }

    rewind(infile);
    return variety;  //  return variety
}
```

그림 7 count_variety op1 바꾼 부분

for loop 내에서 loop의 condition을 확인할 때마다 count_line을 호출하고 있다. 따라서 그림 7과 같이 count_line을 line으로 변경하고 count_line의 값을 받기 위해 infile을 처음으로 되돌리는 rewind 함수를 없앴다.

2） profiling 결과

```
dw@dw-VirtualBox:/media/sf_2020/SystemProgram/HW3/op 1$ gprof op1
Flat profile:

Each sample counts as 0.01 seconds.
 no time accumulated

  %   cumulative   self              self     total
 time   seconds   seconds    calls  Ts/call  Ts/call  name
  0.00      0.00     0.00      195     0.00     0.00  count_variety
  0.00      0.00     0.00        2     0.00     0.00  calculate_momentum
  0.00      0.00     0.00        2     0.00     0.00  count_line
  0.00      0.00     0.00        1     0.00     0.00  calculate_profit
  0.00      0.00     0.00        1     0.00     0.00  data_selection
  0.00      0.00     0.00        1     0.00     0.00  get_user_input
  0.00      0.00     0.00        1     0.00     0.00  make_output_file
  0.00      0.00     0.00        1     0.00     0.00  print_row_data
```

그림 8 optimization 1 time

```
                     Call graph (explanation follows)


granularity: each sample hit covers 2 byte(s) no time propagated

index % time    self  children    called     name
                0.00    0.00       1/195         data_selection [5]
                0.00    0.00       1/195         print_row_data [8]
                0.00    0.00       2/195         calculate_momentum [2]
                0.00    0.00       2/195         main [14]
                0.00    0.00      42/195         make_output_file [7]
                0.00    0.00     147/195         calculate_profit [4]
[1]      0.0    0.00    0.00     195         count_variety [1]
-----------------------------------------------
                0.00    0.00       2/2           get_user_input [6]
[2]      0.0    0.00    0.00       2         calculate_momentum [2]
                0.00    0.00       2/195         count_variety [1]
-----------------------------------------------
                0.00    0.00       1/2           print_row_data [8]
                0.00    0.00       1/2           main [14]
[3]      0.0    0.00    0.00       2         count_line [3]
-----------------------------------------------
                0.00    0.00       1/1           get_user_input [6]
[4]      0.0    0.00    0.00       1         calculate_profit [4]
                0.00    0.00     147/195         count_variety [1]
-----------------------------------------------
                0.00    0.00       1/1           main [14]
[5]      0.0    0.00    0.00       1         data_selection [5]
                0.00    0.00       1/195         count_variety [1]
-----------------------------------------------
                0.00    0.00       1/1           main [14]
[6]      0.0    0.00    0.00       1         get_user_input [6]
                0.00    0.00       2/2           calculate_momentum [2]
                0.00    0.00       1/1           print_row_data [8]
                0.00    0.00       1/1           calculate_profit [4]
                0.00    0.00       1/1           make_output_file [7]
-----------------------------------------------
                0.00    0.00       1/1           get_user_input [6]
[7]      0.0    0.00    0.00       1         make_output_file [7]
                0.00    0.00      42/195         count_variety [1]
-----------------------------------------------
                0.00    0.00       1/1           get_user_input [6]
[8]      0.0    0.00    0.00       1         print_row_data [8]
                0.00    0.00       1/2           count_line [3]
                0.00    0.00       1/195         count_variety [1]
-----------------------------------------------
```

그림 9 optimization 1 call graph

3) profiling 결과 분석

count_variety에서 count_line을 call하지 않게 되었다. 따라서 전체 code에서count_line의 call 횟수가 95747에서 2회로 줄어들었다. 그에 따라서 count_variety를 call하는 함수 모두에서의 run time이 0.14ms에서 측정 단위 이하로 내려간 것을 그림 8을 통해 알 수 있다.

count_variety의 call 횟수는 195로 여전히 높은 것은 것을 확인할 수 있고 그림 9에서 확인해보면 calculate_profit에서 147회, make_output_file에서 42회로 다른 함수에서보다 call하는 횟수가 높은 것을 알 수 있다.

4) 다음 단계 optimization 내용

위에서 확인한 바와 같이 calculate_profit, make_output_file에서 count_vareity의 call 횟수가 많으므로 이 두 함수에서의 call을 줄인다.

C. optimization 2

1) source code

```
// function prototype
int count_line(FILE *infile);
int count_variety(FILE *infile, ROW_DATA *row_data, int line);
void data_selection(FILE *infile, ROW_DATA *row_data, STOCK *select_data, int line);
void calculate_momentum(FILE *infile, ROW_DATA *row_data, STOCK *select_data,
                        int buy_date, int line);
void calculate_profit(FILE *infile, ROW_DATA *row_data, STOCK *select_data,
                      int buy_rank, double min_profit, int buy_date, char work, int line);
void print_row_data(FILE *infile, ROW_DATA *row_data);
void print_the_data(FILE *infile, ROW_DATA *row_data, STOCK *select_data, int print_stock, int line);
void make_output_file(FILE *infile, ROW_DATA *row_data, FILE *outfile, STOCK *select_data, int line);
void get_user_input(FILE *infile, int *buy_date, int *buy_rank, double *min_profit,
                    int *print_stock, ROW_DATA *row_data, STOCK *select_data, FILE *outfile, int line);
```

그림 10 바꿀 function prototype

```
310     for (int k = 1; k <= buy_rank; k++) {
311         for (int i = 0; i < count_variety(infile, row_data, line); i++) {
312             //  variables to get max profit
313             double max_profit = select_data[i].profit[0];
314             //  variable to get max profit's index
315             int max_num;
316             int month = line/count_variety(infile, row_data, line);
317
318             //  calculate profit for stock whose rank is 1 to buy_rank
319             if (select_data[i].rank == k) {
320                 printf("\nrank %d stock no. %d\n", k, select_data[i].stock_no);
321                 printf("date : profit\n\n");
```

그림 11 calculation_profit 바꿀 loop의 condition

```
355     printf("momentum of purchasing date that satisfies minimum\n");
356     for (int i = 0; i < count_variety(infile, row_data, line); i++) {
357         int month = line/count_variety(infile, row_data, line);
358         for (int j = 0; j < month; j++) {
359             if (select_data[i].date[j] == buy_date) {
360                 if (min_profit <= select_data[i].profit[j]) {
361                     printf("profit %d : %d\n", select_data[i].stock_no,
362                         select_data[i].momentum[j]);
363                 }
364             }
365         }
366     }
367
```

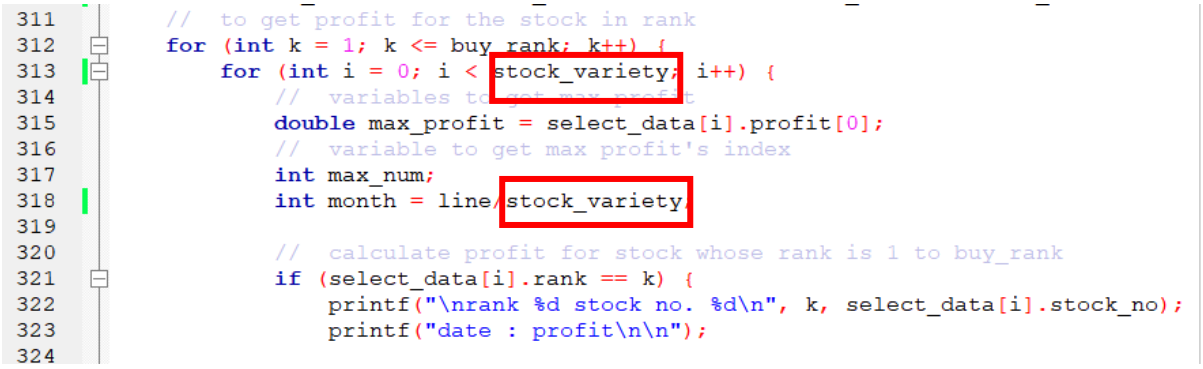
그림 12 calculation_profit 바꿀 loop의 condition2

```
402     //  calculate profit for all stocks
403     for (int i = 0; i < count_variety(infile, row_data, line); i++) {
404         int month = line/count_variety(infile, row_data, line);
405         for (int j = 0; j < month; j++) {
406             select_data[i].profit[j] = (select_data[i].price[0] -
407                                         select_data[i].price[j])/
408                                         (double)select_data[i].price[0];
409         }
410     }
411
412     //  loop for a type of stock
413     for (int i = 0; i < count_variety(infile, row_data, line); i++) {
414         int month = line/count_variety(infile, row_data, line);
415         //  print general information of the stock
416         fprintf(outfile, "%d%c%d%c%d\n", select_data[i].stock_no, ',',
417                     select_data[i].avg_momentum, ',', select_data[i].rank);
418
```

그림 13 make_output_file 바꿀 loop의 condition

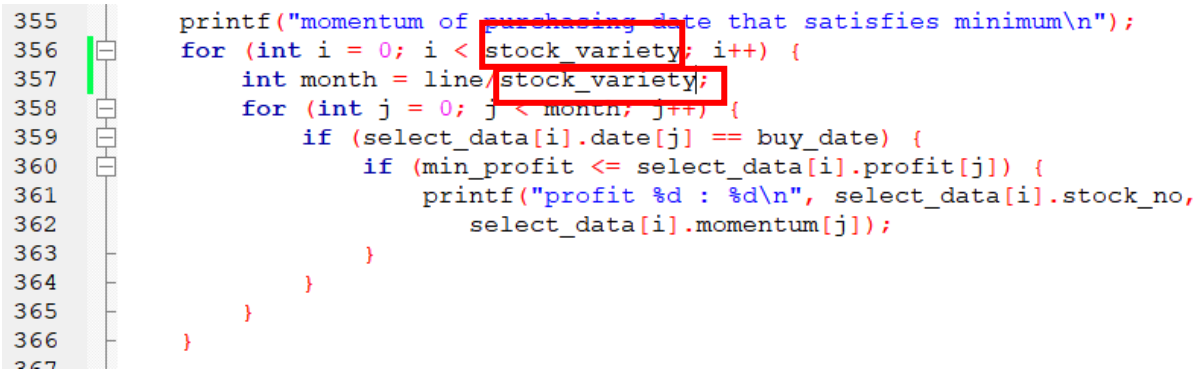
```
51  // function prototype
52  int count_line(FILE *infile);
53  int count_variety(FILE *infile, ROW_DATA *row_data, int line);
54  void data_selection(FILE *infile, ROW_DATA *row_data, STOCK *select_data, int line);
55  void calculate_momentum(FILE *infile, ROW_DATA *row_data, STOCK *select_data,
56                          int buy_date, int line);
57  void calculate_profit(int stock_variety, STOCK *select_data,
58                        int buy_rank, double min_profit, int buy_date, char work, int line);
59  void print_row_data(FILE *infile, ROW_DATA *row_data);
60  void print_the_data(FILE *infile, ROW_DATA *row_data, STOCK *select_data, int print_stock, int line);
61  void make_output_file(int stock_variety, FILE *outfile, STOCK *select_data, int line);
62  void get_user_input(FILE *infile, int *buy_date, int *buy_rank, double *min_profit,
63                      int *print_stock, ROW_DATA *row_data, STOCK *select_data, FILE *outfile, int line);
```

그림 14 바꾼 function prototype



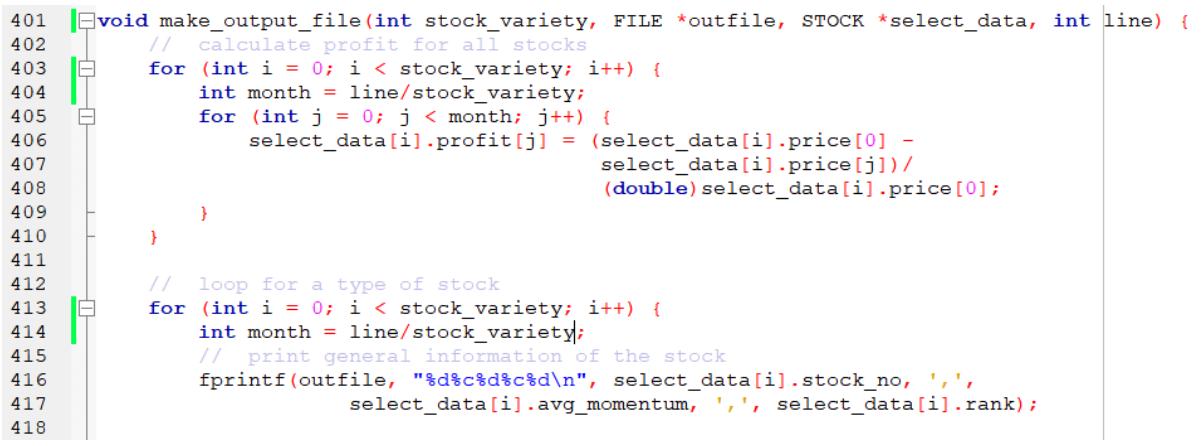
```
311     //  to get profit for the stock in rank
312     for (int k = 1; k <= buy_rank; k++) {
313         for (int i = 0; i < stock_variety; i++) {
314             //  variables to get max profit
315             double max_profit = select_data[i].profit[0];
316             //  variable to get max profit's index
317             int max_num;
318             int month = line/stock_variety;
319
320             //  calculate profit for stock whose rank is 1 to buy_rank
321             if (select_data[i].rank == k) {
322                 printf("\nrank %d stock no. %d\n", k, select_data[i].stock_no);
323                 printf("date : profit\n\n");
324
```

그림 15 calculation_profit 바꾼 loop의 condition



```
355     printf("momentum of purchasing date that satisfies minimum\n");
356     for (int i = 0; i < stock_variety; i++) {
357         int month = line/stock_variety;
358         for (int j = 0; j < month; j++) {
359             if (select_data[i].date[j] == buy_date) {
360                 if (min_profit <= select_data[i].profit[j]) {
361                     printf("profit %d : %d\n", select_data[i].stock_no,
362                         select_data[i].momentum[j]);
363                 }
364             }
365         }
366     }
```

그림 16 calculation_profit 바꾼 loop의 condition2



```
401 void make_output_file(int stock_variety, FILE *outfile, STOCK *select_data, int line) {
402     //  calculate profit for all stocks
403     for (int i = 0; i < stock_variety; i++) {
404         int month = line/stock_variety;
405         for (int j = 0; j < month; j++) {
406             select_data[i].profit[j] = (select_data[i].price[0] -
407                                         select_data[i].price[j])/
408                                         (double)select_data[i].price[0];
409         }
410     }
411
412     //  loop for a type of stock
413     for (int i = 0; i < stock_variety; i++) {
414         int month = line/stock_variety;
415         //  print general information of the stock
416         fprintf(outfile, "%d%c%d%c%d\n", select_data[i].stock_no, ',',
417                     select_data[i].avg_momentum, ',', select_data[i].rank);
418
```

그림 17 make_output_file 바뀐 loop의 condition

그림 10과 14에서 볼 수 있듯이 function에서 넘겨주는 값 자체를 count_variety의 output을 인가받은 stock_vareity로 변경하였다. 이 덕분에 count_variety의 call 횟수도 줄어들었을 뿐 아니라 함수에 넘겨주는 변수 개수도 줄었기 때문에 메모리 측면에서도 이득을 볼 수 있는 변화이다.

2） profiling 결과

```
dw@dw-VirtualBox:/media/sf_2020/SystemProgram/HW3/op 2$ gprof op2_1
Flat profile:

Each sample counts as 0.01 seconds.
 no time accumulated

  %   cumulative   self              self     total
 time   seconds   seconds    calls  Ts/call  Ts/call  name
  0.00      0.00     0.00        7     0.00     0.00  count_variety
  0.00      0.00     0.00        2     0.00     0.00  calculate_momentum
  0.00      0.00     0.00        2     0.00     0.00  count_line
  0.00      0.00     0.00        1     0.00     0.00  calculate_profit
  0.00      0.00     0.00        1     0.00     0.00  data_selection
  0.00      0.00     0.00        1     0.00     0.00  get_user_input
  0.00      0.00     0.00        1     0.00     0.00  make_output_file
  0.00      0.00     0.00        1     0.00     0.00  print_row_data
```

그림 18 op2 후 profiling 결과

```
index % time    self  children    called     name
                0.00    0.00       1/7           data_selection [5]
                0.00    0.00       1/7           print_row_data [8]
                0.00    0.00       1/7           get_user_input [6]
                0.00    0.00       2/7           calculate_momentum [2]
                0.00    0.00       2/7           main [14]
[1]      0.0    0.00    0.00       7         count_variety [1]
-----------------------------------------------
                0.00    0.00       2/2           get_user_input [6]
[2]      0.0    0.00    0.00       2         calculate_momentum [2]
                0.00    0.00       2/7           count_variety [1]
-----------------------------------------------
                0.00    0.00       1/2           print_row_data [8]
                0.00    0.00       1/2           main [14]
[3]      0.0    0.00    0.00       2         count_line [3]
-----------------------------------------------
                0.00    0.00       1/1           get_user_input [6]
[4]      0.0    0.00    0.00       1         calculate_profit [4]
-----------------------------------------------
                0.00    0.00       1/1           main [14]
[5]      0.0    0.00    0.00       1         data_selection [5]
                0.00    0.00       1/7           count_variety [1]
-----------------------------------------------
                0.00    0.00       1/1           main [14]
[6]      0.0    0.00    0.00       1         get_user_input [6]
                0.00    0.00       2/2           calculate_momentum [2]
                0.00    0.00       1/7           count_variety [1]
                0.00    0.00       1/1           print_row_data [8]
                0.00    0.00       1/1           calculate_profit [4]
                0.00    0.00       1/1           make_output_file [7]
-----------------------------------------------
                0.00    0.00       1/1           get_user_input [6]
[7]      0.0    0.00    0.00       1         make_output_file [7]
-----------------------------------------------
                0.00    0.00       1/1           get_user_input [6]
[8]      0.0    0.00    0.00       1         print_row_data [8]
                0.00    0.00       1/2           count_line [3]
                0.00    0.00       1/7           count_variety [1]
-----------------------------------------------
```

그림 19 op2 후 profiling call graph

3) profiling 결과 분석

optimization 결과 count_variety의 call 횟수가 195회에서 7회로 변했다. optimization 1부터 'gprof'의 시간 단위 이하로 running time이 내려갔기 때문에 call 횟수를 중점적으로 code를 수정한다. 여전히 count_variety의 call 횟수가 7회이기 때문에 이를 더 줄일 수 있는 방법을 찾는데 집중할 예정이다.

4) 다음 단계 optimization 내용

그림 19를 통해 count_variety를 대부분의 함수에서 사용하고 있는 것을 알 수 있다. 따라서 count_variety를 이용하는 함수들(data_selection, print_row_data, get_user_input, calculate_momentum, main)을 최초로 call 하는 함수에서 (그림 19에서 확인한 결과 get_user_input이 유력하다.) 한 번만 call하고 이 값을 변수에 할당하여 각 함수에 넘겨주는 방법을 사용한다. 같은 방식으로 call 횟수가 2회인 calculate_momentum과 count_line도 수정할 수 있다면 수정한다.

D. optimization 3

1) source code

```
66  int main(void) {
67      //  pointer to handle files
68      FILE *infile = fopen("all_month_data.csv", "r");  //  input file
69      FILE *outfile = fopen("final_data.csv", "w");  //  output file
70
71      //  check file opening succeed
72      if(infile == NULL || outfile == NULL){
73          printf("file opening failed\n");
74          return 1;
75      }
76
77      int line = count_line(infile);  //  whole line of file, same as data's no.
78      printf("line : %d\n", line);
79
80
81      //  pointer array to save data from infile
82      ROW_DATA *row_data = (ROW_DATA *)malloc(sizeof(ROW_DATA) * line);
83      printf("stock variety : %d\n", count_variety(infile, row_data, line));
84
```

그림 20 바꾸기 전 main 함수

```
80
81      //  pointer array to save data from infile
82      ROW_DATA *row_data = (ROW_DATA *)malloc(sizeof(ROW_DATA) * line);
83      int stock_variety = count_variety(infile, row_data, line);
84      printf("stock variety : %d\n", stock_variety);
85
86      //  pointer array to save collected data
87      STOCK *select_data = (STOCK *)malloc(sizeof(STOCK)*stock_variety);
88
```

그림 21 stock_variety 변수를 만들어 output 인가

```
51  // function prototype
52  int count_line(FILE *infile);
53  int count_variety(FILE *infile, ROW_DATA *row_data, int line);
54  void data_selection(FILE *infile, ROW_DATA *row_data, STOCK *select_data, int line);
55  void calculate_momentum(FILE *infile, ROW_DATA *row_data, STOCK *select_data,
56                          int buy_date, int line);
57  void calculate_profit(int stock_variety, STOCK *select_data,
58                        int buy_rank, double min_profit, int buy_date, char work, int line);
59  void print_row_data(FILE *infile, ROW_DATA *row_data);
60  void print_the_data(FILE *infile, ROW_DATA *row_data, STOCK *select_data, int print_stock, int line);
61  void make_output_file(int stock_variety, FILE *outfile, STOCK *select_data, int line);
62  void get_user_input(FILE *infile, int *buy_date, int *buy_rank, double *min_profit,
63                      int *print_stock, ROW_DATA *row_data, STOCK *select_data, FILE *outfile, int line);
```

그림 22 기존 function prototype

```
51  // function prototype
52  int count_line(FILE *infile);
53  int count_variety(FILE *infile, ROW_DATA *row_data, int line);
54  void data_selection(int stock_variety, ROW_DATA *row_data, STOCK *select_data, int line);
55  void calculate_momentum(int stock_variety, STOCK *select_data,
56                          int buy_date, int line);
57  void calculate_profit(int stock_variety, STOCK *select_data,
58                        int buy_rank, double min_profit, int buy_date, char work, int line);
59  void print_row_data(FILE *infile, ROW_DATA *row_data);
60  void print_the_data(int stock_variety, STOCK *select_data, int print_stock, int line);
61  void make_output_file(int stock_variety, FILE *outfile, STOCK *select_data, int line);
62  void get_user_input(FILE *infile, int *buy_date, int *buy_rank, double *min_profit,
63                      int *print_stock, ROW_DATA *row_data, STOCK *select_data, FILE *outfile, int line);
```

그림 23 stock_variety를 넘겨주도록 function prototype 변경

```
259 void data_selection(FILE *infile, ROW_DATA *row_data, STOCK *select_data, int line) {
260     int stock_variety = count_variety(infile, row_data, line);
261     for (int i = 0; i < stock_variety; i++) {
262         int month = line/stock_variety;
263         for (int j = 0; j < month*stock_variety; j++) {
264             select_data[i].stock_no = row_data[month*i].a_stock_no;
265
266             for (int k = 0; k < month; k++) {
267                 select_data[i].date[k] = row_data[month*i + k].a_date;
268                 select_data[i].price[k] = row_data[month*i + k].a_price;
269                 select_data[i].rank = 1;  //  initialize rank as 1
270                 select_data[i].momentum[k] = 0;  //  initialize momentum as 0
271                 select_data[i].profit[k] = 0;  //  initialize profit as 0
272             }
273         }
274     }
275 }
```

그림 24 count_variety를 함수 내부에서 call

```
259 void data_selection(int stock_variety, STOCK *select_data, int line) {
260     for (int i = 0; i < stock_variety; i++) {
261         int month = line/stock_variety;
262         for (int j = 0; j < month*stock_variety; j++) {
263             select_data[i].stock_no = row_data[month*i].a_stock_no;
264 
265             for (int k = 0; k < month; k++) {
266                 select_data[i].date[k] = row_data[month*i + k].a_date;
267                 select_data[i].price[k] = row_data[month*i + k].a_price;
268                 select_data[i].rank = 1;  //  initialize rank as 1
269                 select_data[i].momentum[k] = 0;  //  initialize momentum as 0
270                 select_data[i].profit[k] = 0;  //  initialize profit as 0
271             }
272         }
273     }
274 }
```

그림 25 data_selection 내부 변경: 넘겨받은 stock_variety 사용

```
276 void calculate_momentum(FILE *infile, ROW_DATA *row_data, STOCK *select_data, in
277     int total = 0;  //  initialize total to calculate average
278     int stock_variety = count_variety(infile, row_data, line);
279 
280     for (int i = 0; i < stock_variety; i++) {
281         int month = line/stock_variety;
282         for (int j = 0; j < month -12; j++) {
283             select_data[i].momentum[j] = select_data[i].price[j] -
284                                          select_data[i].price[j + 12];
285         }
286     }
287 
288     for (int i = 0; i < stock_variety; i++) {
289         int month = line/stock_variety;
290         for (int j = 0; j < month; j++) {
291             if (select_data[i].date[j] == buy_date) {
292                 for (int k = 0; k < 12; k++) {
293                     total += select_data[i].momentum[j + k];
294                 }
```

그림 26 count_variety를 함수 내부에서 call

```
276 void calculate_momentum(int stock_variety, STOCK *select_data, int buy_date, int line) {
277     int total = 0;  //  initialize total to calculate average
278
279     for (int i = 0; i < stock_variety; i++) {
280         int month = line/stock_variety;
281         for (int j = 0; j < month -12; j++) {
282             select_data[i].momentum[j] = select_data[i].price[j] -
283                                          select_data[i].price[j + 12];
284         }
285     }
286
287     for (int i = 0; i < stock_variety; i++) {
288         int month = line/stock_variety;
289         for (int j = 0; j < month; j++) {
290             if (select_data[i].date[j] == buy_date) {
291                 for (int k = 0; k < 12; k++) {
292                     total += select_data[i].momentum[j + k];
293                 }
294             }
295         }
296         select_data[i].avg_momentum = (double)total/12;
297         total = 0;
298     }
299
300     for (int i = 0; i < stock_variety; i++) {
301         for (int j = 0; j < stock_variety; j++) {
302             if (select_data[i].avg_momentum < select_data[j].avg_momentum) {
303                 select_data[i].rank++;
304             }
305         }
306     }
307 }
```

그림 27 calculate_momentum 내부 변경: 넘겨받은 stock_variety 사용

```
382 void print_the_data(FILE *infile, ROW_DATA *row_data, STOCK *select_data, int print_stock, int line) {
383     for (int i = 0; i < count_variety(infile, row_data, line); i++) {
384         if (select_data[i].stock_no == print_stock) {
385             //  print title(average momentum, ranking of stock)
386             printf("average momentum : %d, rank : %d\n",
387                    select_data[i].avg_momentum, select_data[i].rank);
388             //  print name of the data
389             printf("date\tprice\n");
390             int month = line/count_variety(infile, row_data, line);
391             for (int j = 0; j < month; j++) {
392                 //  print date and price of the data
393                 printf("%d\t%d\n", select_data[i].date[j],
394                        select_data[i].price[j]);
395             }
396         }
397     }
398 }
```

그림 28 count_variety를 loop의 condition에서 call

```
382 void print_the_data(int stock_variety, STOCK *select_data, int print_stock, int line) {
383     for (int i = 0; i < stock_variety; i++) {
384         if (select_data[i].stock_no == print_stock) {
385             //  print title(average momentum, ranking of stock)
386             printf("average momentum : %d, rank : %d\n",
387                    select_data[i].avg_momentum, select_data[i].rank);
388             //  print name of the data
389             printf("date\tprice\n");
390             int month = line/stock_variety;
391             for (int j = 0; j < month; j++) {
392                 //  print date and price of the data
393                 printf("%d\t%d\n", select_data[i].date[j],
394                        select_data[i].price[j]);
395             }
396         }
397     }
398 }
```

그림 29 print_the_data 내부 변경: 넘겨받은 stock_variety 사용

```
372  void print_row_data(FILE *infile, ROW_DATA *row_data) {
373      printf("stock\tdate\tprice\n\n");
374      int line = count_line(infile);
375      int month = line/count_variety(infile, row_data, line);
376      for (int i = 0; i < line; i++) {
377          printf("%d\t%d\t%d\n", row_data[i].a_stock_no, row_data[i].a_date,
378                  row_data[i].a_price);
379      }
380  }
372  void print_row_data(FILE *infile, ROW_DATA *row_data, int line) {
373      printf("stock\tdate\tprice\n\n");
374      for (int i = 0; i < line; i++) {
375          printf("%d\t%d\t%d\n", row_data[i].a_stock_no, row_data[i].a_date,
376                  row_data[i].a_price);
377      }
378  }
```

그림 30 data_selection 내부 변경: 사용되지 않는 month 삭제 및 이에 따른 count_variety call 삭제, 넘겨 받은 line 값 사용

```
118  void get_user_input(FILE *infile, int *buy_date, int *buy_rank, double *min_profit,
119                      int *print_stock, ROW_DATA *row_data, STOCK *select_data, FILE *outfile, int line) {
120      char work = '2';
121      int stock_variety = count_variety(infile, row_data, line);
122
123      while (1) {
```

그림 31 get_user_input에서 stock_variety에 count_variety output 저장

그림 20부터 25에서 볼 수 있듯이 각각의 함수에서 count_variety를 호출하던 것을 main과 get_user_input에서 각각 return값을 stock_variety에 저장하여 다른 함수를 호출할 때 사용하고 있다. get_user_input 또한 main에서 stock_variety의 값을 넘겨받아도 프로그램이 구동되는 것에는 문제가 없지만 그림 26에서 볼 수 있듯이 이미 get_user_input이 넘겨 받는 값이 9개로 적정 개수를 초과한 것을 알 수 있다. 따라서 다른 함수 구동을 위해서 필수적으로 infile, row_data, line을 넘겨받아야 하는 get_user_input의 특성 상 count_variety를 함수 내부에서 계산하는 것이 더 효율적일 것이라 판단하여 get_user_input은 변경하지 않았다.

2） profiling 결과

```
dw@dw-VirtualBox:/media/sf_2020/SystemProgram/HW3/op 3$ gprof op3
Flat profile:

Each sample counts as 0.01 seconds.
 no time accumulated

  %   cumulative   self              self     total
 time   seconds   seconds    calls  Ts/call  Ts/call  name
  0.00      0.00     0.00        2     0.00     0.00  calculate_momentum
  0.00      0.00     0.00        2     0.00     0.00  count_variety
  0.00      0.00     0.00        1     0.00     0.00  calculate_profit
  0.00      0.00     0.00        1     0.00     0.00  count_line
  0.00      0.00     0.00        1     0.00     0.00  data_selection
  0.00      0.00     0.00        1     0.00     0.00  get_user_input
  0.00      0.00     0.00        1     0.00     0.00  make_output_file
  0.00      0.00     0.00        1     0.00     0.00  print_row_data
```

그림 32 op3 optimization 결과

```
                     Call graph (explanation follows)


granularity: each sample hit covers 2 byte(s) no time propagated

index % time    self  children    called     name
                0.00    0.00       2/2           get_user_input [6]
[1]      0.0    0.00    0.00       2         calculate_momentum [1]
-----------------------------------------------
                0.00    0.00       1/2           get_user_input [6]
                0.00    0.00       1/2           main [14]
[2]      0.0    0.00    0.00       2         count_variety [2]
-----------------------------------------------
                0.00    0.00       1/1           get_user_input [6]
[3]      0.0    0.00    0.00       1         calculate_profit [3]
-----------------------------------------------
                0.00    0.00       1/1           main [14]
[4]      0.0    0.00    0.00       1         count_line [4]
-----------------------------------------------
                0.00    0.00       1/1           main [14]
[5]      0.0    0.00    0.00       1         data_selection [5]
-----------------------------------------------
                0.00    0.00       1/1           main [14]
[6]      0.0    0.00    0.00       1         get_user_input [6]
                0.00    0.00       2/2           calculate_momentum [1]
                0.00    0.00       1/2           count_variety [2]
                0.00    0.00       1/1           print_row_data [8]
                0.00    0.00       1/1           calculate_profit [3]
                0.00    0.00       1/1           make_output_file [7]
-----------------------------------------------
                0.00    0.00       1/1           get_user_input [6]
[7]      0.0    0.00    0.00       1         make_output_file [7]
-----------------------------------------------
                0.00    0.00       1/1           get_user_input [6]
[8]      0.0    0.00    0.00       1         print_row_data [8]
-----------------------------------------------
```

그림 33 op3 call graph

3) profiling 결과 분석

calculate_variety의 call이 7번에서 main에서 한 번 위에서 설명된 대로 효율성의 문제로 get_user_input에서 한 번 총 2번으로 줄었다. 그림 25에서와 같이 data_selection에서 call되던 count_line을 값을 넘겨 받도록 하여 call 횟수를 1번을 줄였다. calculate_momentum에서 처리하는 값의 경우 주식을 구매하는 날에 따라 값이 달라지는데 이 때문에 프로그램을 종료할 때 생성하는 csv 파일에 담기는 값(all_stock_data에 담긴 data 기준 제일 최근 날짜를 기준으로 1년 동안의 모멘텀 계산)과 프로그램 실행 중 사용하는 값(프로그램 실행 중 값: 입력한 날짜를 기준으로 1년 동안의 모멘텀 계산)이 다르기 때문에 불가피하게 두 번 call해야 한다.

4) 다음 단계 optimization 내용

call 횟수는 output이 바뀌지 않거나 효율성에 문제가 있지 않는 한 최대한 줄인 단계로 보인다. 다음 단계에서는 loop 내부에서 변경되지 않는 값에 대해 중복적인 계산이 있는지 확인한다.

E. optimization 4

1) source code

```
259 void data_selection(int stock_variety, ROW_DATA *row_data, STOCK *select_data, int line) {
260     for (int i = 0; i < stock_variety; i++) {
261         int month = line/stock_variety;
262         for (int j = 0; j < month*stock_variety; j++) {
263             select_data[i].stock_no = row_data[month*i].a_stock_no;
264 
265             for (int k = 0; k < month; k++) {
266                 select_data[i].date[k] = row_data[month*i + k].a_date;
267                 select_data[i].price[k] = row_data[month*i + k].a_price;
268                 select_data[i].rank = 1;  //  initialize rank as 1
269                 select_data[i].momentum[k] = 0;  //  initialize momentum as 0
270                 select_data[i].profit[k] = 0;  //  initialize profit as 0
271             }
272         }
273     }
274 }
```

그림 34 data_selection 변경 전

```
259 void data_selection(int stock_variety, ROW_DATA *row_data, STOCK *select_data, int line, int month) {
260     for (int i = 0; i < stock_variety; i++) {
261         for (int j = 0; j < month*stock_variety; j++) {
262             select_data[i].stock_no = row_data[month*i].a_stock_no;
263 
264             for (int k = 0; k < month; k++) {
265                 select_data[i].date[k] = row_data[month*i + k].a_date;
266                 select_data[i].price[k] = row_data[month*i + k].a_price;
267                 select_data[i].rank = 1;  //  initialize rank as 1
268                 select_data[i].momentum[k] = 0;  //  initialize momentum as 0
269                 select_data[i].profit[k] = 0;  //  initialize profit as 0
270             }
271         }
272     }
273 }
```

그림 35 data_selection loop 내 변수 정의 없앰, month 값 받아서 사용

```
275 void calculate_momentum(int stock_variety, STOCK *select_data, int buy_date, int line) {
276     int total = 0;  //  initialize total to calculate average
277 
278     for (int i = 0; i < stock_variety; i++) {
279         int month = line/stock_variety;
280         for (int j = 0; j < month -12; j++) {
281             select_data[i].momentum[j] = select_data[i].price[j] -
282                                          select_data[i].price[j + 12];
283         }
284     }
285 
286     for (int i = 0; i < stock_variety; i++) {
287         int month = line/stock_variety;
288         for (int j = 0; j < month; j++) {
289             if (select_data[i].date[j] == buy_date) {
290                 for (int k = 0; k < 12; k++) {
291                     total += select_data[i].momentum[j + k];
292                 }
293             }
294         }
295         select_data[i].avg_momentum = (double)total/12;
296         total = 0;
297     }
298 
299     for (int i = 0; i < stock_variety; i++) {
300         for (int j = 0; j < stock_variety; j++) {
301             if (select_data[i].avg_momentum < select_data[j].avg_momentum) {
302                 select_data[i].rank++;
303             }
304         }
305     }
306 }
```

그림 36 calculate_momentum 변경 전

```
275 void calculate_momentum(int stock_variety, STOCK *select_data, int buy_date, int line, int month) {
276     int total = 0;  //  initialize total to calculate average
277 
278     for (int i = 0; i < stock_variety; i++) {
279         for (int j = 0; j < month -12; j++) {
280             select_data[i].momentum[j] = select_data[i].price[j] -
281                                          select_data[i].price[j + 12];
282         }
283     }
284 
285     for (int i = 0; i < stock_variety; i++) {
286         for (int j = 0; j < month; j++) {
287             if (select_data[i].date[j] == buy_date) {
288                 for (int k = 0; k < 12; k++) {
289                     total += select_data[i].momentum[j + k];
290                 }
291             }
292         }
293         select_data[i].avg_momentum = (double)total/12;
294         total = 0;
295     }
296 
297     for (int i = 0; i < stock_variety; i++) {
298         for (int j = 0; j < stock_variety; j++) {
299             if (select_data[i].avg_momentum < select_data[j].avg_momentum) {
300                 select_data[i].rank++;
301             }
302         }
303     }
304 }
```

그림 37 calculate_momentum loop 내 변수 정의 없앰, month 값 받아서 사용

```
306  void calculate_profit(int stock_variety, STOCK *select_data, int buy_rank,
307                        double min_profit, int buy_date, char work, int line) {
308      //  to get profit for the stock in rank
309      for (int k = 1; k <= buy_rank; k++) {
310          for (int i = 0; i < stock_variety; i++) {
311              //  variables to get max profit
312              double max_profit = select_data[i].profit[0];
313              //  variable to get max profit's index
314              int max_num;
315              int month = line/stock_variety;
316
317              //  calculate profit for stock whose rank is 1 to buy_rank
318              if (select_data[i].rank == k) {
319                  printf("\nrank %d stock no. %d\n", k, select_data[i].stock_no);
320                  printf("date : profit\n\n");
321
322                  for (int j = 0; j < month; j++) {
323                      select_data[i].profit[j] = (select_data[i].price[0] -
324                                                  select_data[i].price[j])/
325                                                  (double)select_data[i].price[j];
326                      //  print the profit
327                      printf("%d : %lf\n", select_data[i].date[j],
328                                 select_data[i].profit[j]);
329
330                  }
331
332                  //  find max profit
333                  for (int j = 0; j < month; j++) {
334                      if (max_profit < select_data[i].profit[j]) {
335                          max_profit = select_data[i].profit[j];
336                          max_num = j;
337                      }
338
```

그림 38 calculate_profit 변경 전1

```
352      printf("momentum of purchasing date that satisfies minimum\n");
353      for (int i = 0; i < stock_variety; i++) {
354          int month = line/stock_variety;
355          for (int j = 0; j < month; j++) {
356              if (select_data[i].date[j] == buy_date) {
357                  if (min_profit <= select_data[i].profit[j]) {
358                      printf("profit %d : %d\n", select_data[i].stock_no,
359                         select_data[i].momentum[j]);
360                  }
361              }
362          }
363      }
364
365      //  mark finishing find stocks that satisfy minimum profit
366      printf("\nfinished finding stocks\n\n");
367  }
```

그림 39 calculate_profit 변경 전2

```
306  void calculate_profit(int stock_variety, STOCK *select_data, int buy_rank,
307                        double min_profit, int buy_date, char work, int line) {
308      int month = line/stock_variety;
309      //  to get profit for the stock in rank
310      for (int k = 1; k <= buy_rank; k++) {
311          for (int i = 0; i < stock_variety; i++) {
312              //  variables to get max profit
313              double max_profit = select_data[i].profit[0];
314              //  variable to get max profit's index
315              int max_num;
316
317              //  calculate profit for stock whose rank is 1 to buy_rank
318              if (select_data[i].rank == k) {
319                  printf("\nrank %d stock no. %d\n", k, select_data[i].stock_no);
320                  printf("date : profit\n\n");
321
322                  for (int j = 0; j < month; j++) {
323                      select_data[i].profit[j] = (select_data[i].price[0] -
324                                                  select_data[i].price[j])/
325                                                  (double)select_data[i].price[j];
326                      //  print the profit
327                      printf("%d : %lf\n", select_data[i].date[j],
328                                 select_data[i].profit[j]);
329                  }
330
331                  //  find max profit
332                  for (int j = 0; j < month; j++) {
333                      if (max_profit < select_data[i].profit[j]) {
334                          max_profit = select_data[i].profit[j];
335                          max_num = j;
336                      }
337
351      printf("momentum of purchasing date that satisfies minimum\n");
352      for (int i = 0; i < stock_variety; i++) {
353          for (int j = 0; j < month; j++) {
354              if (select_data[i].date[j] == buy_date) {
355                  if (min_profit <= select_data[i].profit[j]) {
356                      printf("profit %d : %d\n", select_data[i].stock_no,
357                         select_data[i].momentum[j]);
358                  }
359              }
360          }
361      }
362
363      //  mark finishing find stocks that satisfy minimum profit
364      printf("\nfinished finding stocks\n\n");
365  }
```

그림 40 calculate_profit 내 변수 정의 없앰

```
376  void print_the_data(int stock_variety, STOCK *select_data, int print_stock, int line) {
377      for (int i = 0; i < stock_variety; i++) {
378          if (select_data[i].stock_no == print_stock) {
379              //  print title(average momentum, ranking of stock)
380              printf("average momentum : %d, rank : %d\n",
381                     select_data[i].avg_momentum, select_data[i].rank);
382              //  print name of the data
383              printf("date\tprice\n");
384              int month = line/stock_variety;
385              for (int j = 0; j < month; j++) {
386                  //  print date and price of the data
387                  printf("%d\t%d\n", select_data[i].date[j],
388                         select_data[i].price[j]);
389              }
390          }
391      }
392  }
```

그림 41 print_the_data 변경 전

```
376 void print_the_data(int stock_variety, STOCK *select_data, int print_stock, int line, int month) {
377     for (int i = 0; i < stock_variety; i++) {
378         if (select_data[i].stock_no == print_stock) {
379             //  print title(average momentum, ranking of stock)
380             printf("average momentum : %d, rank : %d\n",
381                     select_data[i].avg_momentum, select_data[i].rank);
382             //  print name of the data
383             printf("date\tprice\n");
384             for (int j = 0; j < month; j++) {
385                 //  print date and price of the data
386                 printf("%d\t%d\n", select_data[i].date[j],
387                         select_data[i].price[j]);
388             }
389         }
390     }
391 }
```

그림 42 print_the_data loop 내 변수 정의 없앰, month 값 받아서 사용

```
393 void make_output_file(int stock_variety, FILE *outfile, STOCK *select_data, int line) {
394     //  calculate profit for all stocks
395     for (int i = 0; i < stock_variety; i++) {
396         int month = line/stock_variety;
397         for (int j = 0; j < month; j++) {
398             select_data[i].profit[j] = (select_data[i].price[0] -
399                                         select_data[i].price[j])/
400                                         (double)select_data[i].price[0];
401         }
402     }
403
404     //  loop for a type of stock
405     for (int i = 0; i < stock_variety; i++) {
406         int month = line/stock_variety;
407         //  print general information of the stock
408         fprintf(outfile, "%d%c%d%c%d\n", select_data[i].stock_no, ',',
409                     select_data[i].avg_momentum, ',', select_data[i].rank);
410
411         //  print specific data of the stock
412         for (int j = 0; j < month; j++) {
413             fprintf(outfile, "%d%c%d%c%d%c%lf\n", select_data[i].date[j], ',',
414                     select_data[i].price[j], ',', select_data[i].momentum[j],
415                     ',', select_data[i].profit[j]);
416         }
417     }
418 }
```

그림 43 make_output_file 변경 전

```
393 void make_output_file(int stock_variety, FILE *outfile, STOCK *select_data, int line, int month) {
394     //  calculate profit for all stocks
395     for (int i = 0; i < stock_variety; i++) {
396         for (int j = 0; j < month; j++) {
397             select_data[i].profit[j] = (select_data[i].price[0] -
398                                         select_data[i].price[j])/
399                                         (double)select_data[i].price[0];
400         }
401     }
402
403     //  loop for a type of stock
404     for (int i = 0; i < stock_variety; i++) {
405         //  print general information of the stock
406         fprintf(outfile, "%d%c%d%c%d\n", select_data[i].stock_no, ',',
407                     select_data[i].avg_momentum, ',', select_data[i].rank);
408
409         //  print specific data of the stock
410         for (int j = 0; j < month; j++) {
411             fprintf(outfile, "%d%c%d%c%d%c%lf\n", select_data[i].date[j], ',',
412                     select_data[i].price[j], ',', select_data[i].momentum[j],
413                     ',', select_data[i].profit[j]);
414         }
415     }
416 }
```

그림 44 make_output_file loop 내 변수 정의 없앰, month 값 받아서 사용

```c
int main(void) {
    //  pointer to handle files
    FILE *infile = fopen("all_month_data.csv", "r");  //  input file
    FILE *outfile = fopen("final_data.csv", "w");  //  output file

    //  check file opening succeed
    if(infile == NULL || outfile == NULL){
        printf("file opening failed\n");
        return 1;
    }

    int line = count_line(infile);  //  whole line of file, same as data's no.
    printf("line : %d\n", line);


    //  pointer array to save data from infile
    ROW_DATA *row_data = (ROW_DATA *)malloc(sizeof(ROW_DATA) * line);
    int stock_variety = count_variety(infile, row_data, line);
    printf("stock variety : %d\n", stock_variety);

    int month = line/stock_variety;

    //  pointer array to save collected data
    STOCK *select_data = (STOCK *)malloc(sizeof(STOCK)*stock_variety);

    //  check if dynamic memory allocation is succeed
```

```c
void get_user_input(FILE *infile, int *buy_date, int *buy_rank, double *min_profit,
                    int *print_stock, ROW_DATA *row_data, STOCK *select_data, FILE *outfile, int line) {
    char work = '2';
    int stock_variety = count_variety(infile, row_data, line);
    int month = line/stock_variety;

    while (1) {
        if (work == '1') {
            //  print the data of row_data
            print_row_data(infile, row_data, line);

            //  repeat selecting mode
            printf("what to do?\n");
            printf("1. print row data \n");
            printf("2.buy stock, find momentum that gives minimum profit\n");
            printf("3.print data of specific stock\n F: terminate");
            scanf(" %c", &work);
        } else if (work == '2') {
            printf("enter the day you want to buy stocks(20150514 ~ 20190514)\n");
            scanf("%d", buy_date);

            while ((((*buy_date - 14) % 100) != 0) || (*buy_date < 20150513)
```

그림 45 main, get_user_input에서 변수 month 선언

loop 내부를 살펴본 결과 값이 바뀌지 않는 변수 month가 loop 내에서 선언되고 값을 할당 받고 있었다. 이 경우 loop을 돌 때마다 변수를 생성하고 값을 선언하기 때문에 run time에 영향을 줄 수 있다. month의 경우 나누기 연산이었기 때문에 영향이 다른 operator보다 더 큼에도 불구하고 불필요하게 여러 번 계산되고 있는 것을 개선하기 위해 코드를 변경하였다. 변수의 선언을 loop 밖에서 하거나 함수의 input(기존에 input 개수가 많지 않을 때)으로 받아와 사용하였다. loop 내에서 값이 변화하는 변수라 하더라도 메모리 할당 측면에서 loop의 바깥에 선언을 하는 것이 더 효율적이기 때문에 할 수 있다면 선언은 loop 바깥에서 미리 해놓는 것이 좋다고 판단하였다.

loop을 살펴보면서 수업 시간에 배운대로 row_major_order로 저장된 값을 접근하

는지도 살펴보았는데 다행히 적절하게 접근하고 있었다. 이는 메모리의 locality 측면에서 많은 도움이 될 것이다.

2） profiling 결과

```
dw@dw-VirtualBox:/media/sf_2020/SystemProgram/HW3/op 4$ gprof op4
Flat profile:

Each sample counts as 0.01 seconds.
 no time accumulated

  %   cumulative   self              self     total
 time   seconds   seconds    calls  Ts/call  Ts/call  name
  0.00      0.00     0.00        2     0.00     0.00  calculate_momentum
  0.00      0.00     0.00        2     0.00     0.00  count_variety
  0.00      0.00     0.00        1     0.00     0.00  calculate_profit
  0.00      0.00     0.00        1     0.00     0.00  count_line
  0.00      0.00     0.00        1     0.00     0.00  data_selection
  0.00      0.00     0.00        1     0.00     0.00  get_user_input
  0.00      0.00     0.00        1     0.00     0.00  make_output_file
  0.00      0.00     0.00        1     0.00     0.00  print_row_data
```

그림 46 op4 profiling result

```
                     Call graph (explanation follows)


granularity: each sample hit covers 2 byte(s) no time propagated

index % time    self  children    called     name
                0.00    0.00       2/2           get_user_input [6]
[1]      0.0    0.00    0.00       2         calculate_momentum [1]
-----------------------------------------------
                0.00    0.00       1/2           get_user_input [6]
                0.00    0.00       1/2           main [14]
[2]      0.0    0.00    0.00       2         count_variety [2]
-----------------------------------------------
                0.00    0.00       1/1           get_user_input [6]
[3]      0.0    0.00    0.00       1         calculate_profit [3]
-----------------------------------------------
                0.00    0.00       1/1           main [14]
[4]      0.0    0.00    0.00       1         count_line [4]
-----------------------------------------------
                0.00    0.00       1/1           main [14]
[5]      0.0    0.00    0.00       1         data_selection [5]
-----------------------------------------------
                0.00    0.00       1/1           main [14]
[6]      0.0    0.00    0.00       1         get_user_input [6]
                0.00    0.00       2/2           calculate_momentum [1]
                0.00    0.00       1/2           count_variety [2]
                0.00    0.00       1/1           print_row_data [8]
                0.00    0.00       1/1           calculate_profit [3]
                0.00    0.00       1/1           make_output_file [7]
-----------------------------------------------
                0.00    0.00       1/1           get_user_input [6]
[7]      0.0    0.00    0.00       1         make_output_file [7]
-----------------------------------------------
                0.00    0.00       1/1           get_user_input [6]
[8]      0.0    0.00    0.00       1         print_row_data [8]
-----------------------------------------------
```

그림 47 op4 call graph

3) profiling 결과 분석

수업시간에 다뤘던 loop 내의 inefficiency는 어느 정도 제거한 상태로 보인다. 또한 cache friendly code를 작성함으로써 memory locality 측면에서도 이점이 있도록 코드를 작성하였다.

실행 시간이 작아 profiling 결과를 보고 다음 optimization을 결정하기 어려워졌다. 따라서 임의로 프로그램에서 명시한 input 파일(all_month_data.csv)과 형식은 같으나 유의미한 값을 갖진 않는 csv(all_stock_data.csv) 파일을 만들어 실행하고 이에 따른 실행 시간을 분석해보기로 하였다. 여기서 쓰인 csv 파일은 총 20794 line을 가진 csv 파일로 앞에서 명시된 대로 주식 번호, 날짜, 가격 순으로 저장되어 있다. 날짜는 계산을 위해 임의로 모두 14일로 변경하였고 년도와 월 정보만 유지시켰다. 따라서 이 input file을 통해 계산된 프로그램의 output은 유의미하다고 할 수 없으며 다만 실행 시간을 측정하기 위해 사용된 것이다.

이 파일을 이용하여 측정한 실행시간은 다음과 같다.

```
dw@dw-VirtualBox:/media/sf_2020/SystemProgram/HW3/op 1$ gprof op1_2
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 97.15      0.66     0.66        1   660.63   660.68  data_selection
  2.94      0.68     0.02      393     0.05     0.05  count_variety
  0.00      0.68     0.00        2     0.00     0.05  calculate_momentum
  0.00      0.68     0.00        2     0.00     0.00  count_line
  0.00      0.68     0.00        1     0.00    15.33  calculate_profit
  0.00      0.68     0.00        1     0.00    19.87  get_user_input
  0.00      0.68     0.00        1     0.00     4.38  make_output_file
  0.00      0.68     0.00        1     0.00     0.05  print_row_data
```

그림 48 큰 input file(all_stock_data.csv) op1 profile

```
dw@dw-VirtualBox:/media/sf_2020/SystemProgram/HW3/op 2$ gprof op2_2
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
100.10      0.66     0.66        1   660.66   660.66  data_selection
  0.00      0.66     0.00        7     0.00     0.00  count_variety
  0.00      0.66     0.00        2     0.00     0.00  calculate_momentum
  0.00      0.66     0.00        2     0.00     0.00  count_line
  0.00      0.66     0.00        1     0.00     0.00  calculate_profit
  0.00      0.66     0.00        1     0.00     0.00  get_user_input
  0.00      0.66     0.00        1     0.00     0.00  make_output_file
  0.00      0.66     0.00        1     0.00     0.00  print_row_data
```

그림 49 큰 input file(all_stock_data.csv) op2 profile

```
dw@dw-VirtualBox:/media/sf_2020/SystemProgram/HW3/op 3$ gprof op3_2
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
100.10      0.66     0.66        1   660.68   660.68  data_selection
  0.00      0.66     0.00        2     0.00     0.00  calculate_momentum
  0.00      0.66     0.00        2     0.00     0.00  count_variety
  0.00      0.66     0.00        1     0.00     0.00  calculate_profit
  0.00      0.66     0.00        1     0.00     0.00  count_line
  0.00      0.66     0.00        1     0.00     0.00  get_user_input
  0.00      0.66     0.00        1     0.00     0.00  make_output_file
  0.00      0.66     0.00        1     0.00     0.00  print_row_data
```

그림 50 큰 input file(all_stock_data.csv) op3 profile

```
dw@dw-VirtualBox:/media/sf_2020/SystemProgram/HW3/op 4$ gprof op4_2
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
100.11      0.66     0.66        1   660.69   660.69  data_selection
  0.00      0.66     0.00        2     0.00     0.00  calculate_momentum
  0.00      0.66     0.00        2     0.00     0.00  count_variety
  0.00      0.66     0.00        1     0.00     0.00  calculate_profit
  0.00      0.66     0.00        1     0.00     0.00  count_line
  0.00      0.66     0.00        1     0.00     0.00  get_user_input
  0.00      0.66     0.00        1     0.00     0.00  make_output_file
  0.00      0.66     0.00        1     0.00     0.00  print_row_data
```

그림 51 큰 input file(all_stock_data.csv) op4 profile

그림 37부터 40까지 결과에서 볼 수 있듯이 call 횟수에 의한 실행시간은 어느 정도 줄여졌지만 data_selection에서 대부분의 시간이 소요되고 있는 것을 알 수 있다.

4) 다음 단계 optimization 내용

profiling 결과를 분석해봤을 때 data_selection이 call 횟수가 1번임에도 많은 시간을 소요하고 있기 때문에 바꿀 수 있는 부분이 있는지 확인해볼 것이다.

기존 data_selection 함수는 파일에서 읽어온 데이터(row_data에 저장되어 있다.)를 가공할 수 있는 데이터의 형식으로 분리하여 select_data에 저장하고 rank, momentum, profit을 초기화하는 역할을 했다. 이 때 모든 데이터에 대해 for loop으로 처리를 하기 때문에 수업 시간에 배운 optimization blocker를 발생시키는 요인이 있는지 확인하고 loop unrolling, multiple accumulation, reassociation transformation 등이 가능한지 탐색한다.

F. optimization 5

1) source code

```
262 void data_selection(int stock_variety, ROW_DATA *row_data, STOCK *select_data, int line, int month) {
263     for (int i = 0; i < stock_variety; i++) {
264         for (int j = 0; j < month*stock_variety; j++) {
265             select_data[i].stock_no = row_data[month*i].a_stock_no;
266
267             for (int k = 0; k < month; k++) {
268                 select_data[i].date[k] = row_data[month*i + k].a_date;
269                 select_data[i].price[k] = row_data[month*i + k].a_price;
270                 select_data[i].rank = 1;  //  initialize rank as 1
271                 select_data[i].momentum[k] = 0;  //  initialize momentum as 0
272                 select_data[i].profit[k] = 0;  //  initialize profit as 0
273             }
274         }
275     }
276 }
```

그림 52 op 5 data_selection 바꾸기 전

```
262 void data_selection(int stock_variety, ROW_DATA *row_data, STOCK *select_data, int line, int month) {
263     for (int i = 0; i < stock_variety; i++) {
264         select_data[i].stock_no = row_data[month*i].a_stock_no;
265
266         for (int k = 0; k < month; k++) {
267             select_data[i].date[k] = row_data[month*i + k].a_date;
268             select_data[i].price[k] = row_data[month*i + k].a_price;
269             select_data[i].rank = 1;  //  initialize rank as 1
270             select_data[i].momentum[k] = 0;  //  initialize momentum as 0
271             select_data[i].profit[k] = 0;  //  initialize profit as 0
272         }
273     }
274 }
```

그림 53 op 5 data_selection 바꾼 후

기존 data_selection 함수를 분석해보니 같은 일(빨간 상자 안)을 여러 번 반복하고 있었다.

stock_variety만큼 반복---------- ----line만큼 반복---------- --------line/stock_variety만큼 반복------ →총 반복 횟수 $stock\ variety * line * \frac{line}{stock\ variety} = line^2$	stock_variety만큼 반복---------- ----line/stock_variety만큼 반복------ →총 반복 횟수 $stock\ variety * \frac{line}{stock\ variety} = line$

그림 54 바꾸기 전/후 실행 횟수 비교

그러나 data_selection은 input file의 line만큼만 loop 내부를 반복하며 data를 저장하면 되기 때문에 바꾸기 전과 바꾼 후의 결과는 차이가 없지만 run time에는 큰 영향을 줬을 것이라 예상되어 data_selection 내부의 j에 대한 loop를 제거하였다.

2） profiling 결과

```
dw@dw-VirtualBox:/media/sf_2020/SystemProgram/HW3/op 5$ gprof op5
Flat profile:

Each sample counts as 0.01 seconds.
 no time accumulated

  %   cumulative   self              self     total
 time   seconds   seconds    calls  Ts/call  Ts/call  name
  0.00      0.00     0.00        2     0.00     0.00  calculate_momentum
  0.00      0.00     0.00        2     0.00     0.00  count_variety
  0.00      0.00     0.00        1     0.00     0.00  calculate_profit
  0.00      0.00     0.00        1     0.00     0.00  count_line
  0.00      0.00     0.00        1     0.00     0.00  data_selection
  0.00      0.00     0.00        1     0.00     0.00  get_user_input
  0.00      0.00     0.00        1     0.00     0.00  make_output_file
  0.00      0.00     0.00        1     0.00     0.00  print_row_data
```

그림 55 op5 profiling result

```
                     Call graph (explanation follows)


granularity: each sample hit covers 2 byte(s) no time propagated

index % time    self  children    called     name
                0.00    0.00       2/2           get_user_input [6]
[1]      0.0    0.00    0.00       2         calculate_momentum [1]
-----------------------------------------------
                0.00    0.00       1/2           get_user_input [6]
                0.00    0.00       1/2           main [14]
[2]      0.0    0.00    0.00       2         count_variety [2]
-----------------------------------------------
                0.00    0.00       1/1           get_user_input [6]
[3]      0.0    0.00    0.00       1         calculate_profit [3]
-----------------------------------------------
                0.00    0.00       1/1           main [14]
[4]      0.0    0.00    0.00       1         count_line [4]
-----------------------------------------------
                0.00    0.00       1/1           main [14]
[5]      0.0    0.00    0.00       1         data_selection [5]
-----------------------------------------------
                0.00    0.00       1/1           main [14]
[6]      0.0    0.00    0.00       1         get_user_input [6]
                0.00    0.00       2/2           calculate_momentum [1]
                0.00    0.00       1/2           count_variety [2]
                0.00    0.00       1/1           print_row_data [8]
                0.00    0.00       1/1           calculate_profit [3]
                0.00    0.00       1/1           make_output_file [7]
-----------------------------------------------
                0.00    0.00       1/1           get_user_input [6]
[7]      0.0    0.00    0.00       1         make_output_file [7]
-----------------------------------------------
                0.00    0.00       1/1           get_user_input [6]
[8]      0.0    0.00    0.00       1         print_row_data [8]
-----------------------------------------------
```

그림 56 op 5 call graph

3) profiling 결과 분석

예상대로 run time이 대폭 줄어든 것을 볼 수 있다. 큰 input file의 line 수는 20794로 코드를 바꾸기 전에는 432390436(20794^2)번의 동일한 코드를 반복한 반면 수정 후에는 20794번만 반복한다. 432369642번의 실행이 줄어들게 된 것이다. 그 결과로 data_selection의 실행 시간이 눈에 띄게 줄었고 실행 순서가 긴 순으로 정렬된 그림 44의 profiling result에서도 순위가 1위에서 5위로 내려간 것을 확인할 수 있다.

4) 다음 단계 optimization 내용

optimization 5에서는 data_selection의 run time을 줄이긴 했지만 optimization 4에서 언급한 것처럼 loop unrollling등을 사용하여 줄인 것은 아니다. 다음 단계에서는 for loop의 condition check 횟수 감소 등의 이유로 실행 시간을 줄일 수 있는 loop unrolling을 사용할 예정이다. 그림 44를 토대로 op 5에서 가장 실행 시간이 오래 걸리는 calculate_momentum에 대해서 loop unrolling와 multiple accumulation을 적용해 볼 예정이다. calculate_momentum은 call 횟수도 다른 함수보다 많기 때문에 수정한다면 실행 시간 단축에 영향이 클 것으로 예상된다.

G. optimization 6

1) source code

```
276  void calculate_momentum(int stock_variety, STOCK *select_data, int buy_date, int line, int month) {
277      int total = 0;  //  initialize total to calculate average
278
279      for (int i = 0; i < stock_variety; i++) {
280          for (int j = 0; j < month -12; j++) {
281              select_data[i].momentum[j] = select_data[i].price[j] -
282                                           select_data[i].price[j + 12];
283          }
284      }
285
286      for (int i = 0; i < stock_variety; i++) {
287          for (int j = 0; j < month; j++) {
288              if (select_data[i].date[j] == buy_date) {
289                  for (int k = 0; k < 12; k++) {
290                      total += select_data[i].momentum[j + k];
291                  }
292              }
293          }
294          select_data[i].avg_momentum = (double)total/12;
295          total = 0;
296      }
```

그림 57 op6 진행 전

```
276 void calculate_momentum(int stock_variety, STOCK *select_data, int buy_date, int line, int month) {
277     int total0 = 0;
278     int total1 = 0;  //  initialize total to calculate average
279
280     for (int i = 0; i < stock_variety; i++) {
281         for (int j = 0; j < month -12; j++) {
282             select_data[i].momentum[j] = select_data[i].price[j] -
283                                          select_data[i].price[j + 12];
284         }
285     }
286
287     for (int i = 0; i < stock_variety; i++) {
288         for (int j = 0; j < month; j++) {
289             if (select_data[i].date[j] == buy_date) {
290                 for (int k = 0; k < 12; k+=2) {
291                     total0 += select_data[i].momentum[j + k];
292                     total1 += select_data[i].momentum[j + k +1];
293                 }
294             }
295         }
296         select_data[i].avg_momentum = (double)(total0 + total1)/12;
297         total0 = 0;
298         total1 = 0;
299     }
```

그림 58 2x2 unrolling

(확인 결과 두 code의 output은 동일하다.)

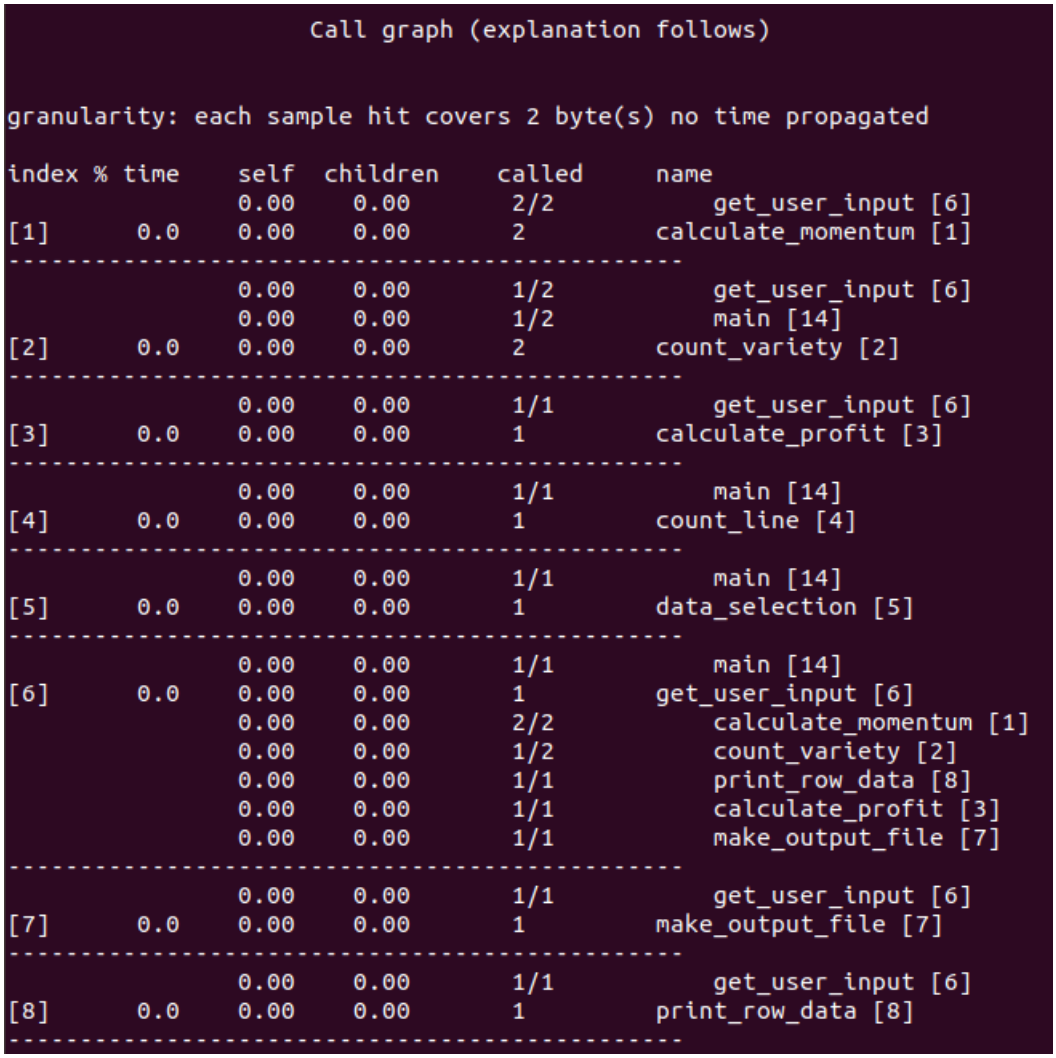
2） profiling 결과

```
dw@dw-VirtualBox:/media/sf_2020/SystemProgram/HW3/op 6$ gprof op6
Flat profile:

Each sample counts as 0.01 seconds.
 no time accumulated

  %   cumulative   self              self     total
 time   seconds   seconds    calls  Ts/call  Ts/call  name
  0.00      0.00     0.00        2     0.00     0.00  calculate_momentum
  0.00      0.00     0.00        2     0.00     0.00  count_variety
  0.00      0.00     0.00        1     0.00     0.00  calculate_profit
  0.00      0.00     0.00        1     0.00     0.00  count_line
  0.00      0.00     0.00        1     0.00     0.00  data_selection
  0.00      0.00     0.00        1     0.00     0.00  get_user_input
  0.00      0.00     0.00        1     0.00     0.00  make_output_file
  0.00      0.00     0.00        1     0.00     0.00  print_row_data
```

그림 59 op6 profiling result



```
                     Call graph (explanation follows)


granularity: each sample hit covers 2 byte(s) no time propagated

index % time    self  children    called     name
                0.00    0.00       2/2           get_user_input [6]
[1]      0.0    0.00    0.00       2         calculate_momentum [1]
-----------------------------------------------
                0.00    0.00       1/2           get_user_input [6]
                0.00    0.00       1/2           main [14]
[2]      0.0    0.00    0.00       2         count_variety [2]
-----------------------------------------------
                0.00    0.00       1/1           get_user_input [6]
[3]      0.0    0.00    0.00       1         calculate_profit [3]
-----------------------------------------------
                0.00    0.00       1/1           main [14]
[4]      0.0    0.00    0.00       1         count_line [4]
-----------------------------------------------
                0.00    0.00       1/1           main [14]
[5]      0.0    0.00    0.00       1         data_selection [5]
-----------------------------------------------
                0.00    0.00       1/1           main [14]
[6]      0.0    0.00    0.00       1         get_user_input [6]
                0.00    0.00       2/2           calculate_momentum [1]
                0.00    0.00       1/2           count_variety [2]
                0.00    0.00       1/1           print_row_data [8]
                0.00    0.00       1/1           calculate_profit [3]
                0.00    0.00       1/1           make_output_file [7]
-----------------------------------------------
                0.00    0.00       1/1           get_user_input [6]
[7]      0.0    0.00    0.00       1         make_output_file [7]
-----------------------------------------------
                0.00    0.00       1/1           get_user_input [6]
[8]      0.0    0.00    0.00       1         print_row_data [8]
-----------------------------------------------
```

그림 60 op6 call graph

3) profiling 결과 분석

calculate_momentum 내의 loop 중에서도 실행 횟수가 높은 것 중 2x2 unrolling이 적용가능한 for loop에 적용해보았다. 코드를 수정했음에도 그림 48에서 보는 것과 같이 순서조차도 바뀌지 않아 혹시 register spill이 일어난 것은 아닌지 확인해보았다.

```
calculate_momentum:
.LFB46:
        .cfi_startproc
        pushq   %rbx
        .cfi_def_cfa_offset 16
        .cfi_offset 3, -16
        movl    $0, %ebx
        jmp     .L25
.L26:
        movslq  %ebx, %rax
        imulq   $19832, %rax, %rax
        addq    %rsi, %rax
        movslq  %ecx, %r11
        movl    3968(%rax,%r11,4), %r9d
        leal    12(%rcx), %r10d
        movslq  %r10d, %r10
        subl    3968(%rax,%r10,4), %r9d
        movl    %r9d, 7932(%rax,%r11,4)
        addl    $1, %ecx
.L27:
        leal    -12(%r8), %eax
        cmpl    %ecx, %eax
        jg      .L26
        addl    $1, %ebx
.L25:
        cmpl    %edi, %ebx
        jge     .L43
        movl    $0, %ecx
        jmp     .L27
.L43:
        movl    $0, %ebx
        jmp     .L28
.L31:
        leal    (%r10,%rax), %ecx
        movslq  %ecx, %rcx
        addl    7932(%r9,%rcx,4), %r11d
        addl    $1, %eax
.L29:
        cmpl    $11, %eax
        jle     .L31
.L30:
        addl    $1, %r10d
.L33:
        cmpl    %r8d, %r10d
        jge     .L44
        movslq  %ebx, %r9
        imulq   $19832, %r9, %r9
        addq    %rsi, %r9
        movslq  %r10d, %rax
        cmpl    %edx, 4(%r9,%rax,4)
        jne     .L30
        movl    $0, %eax
        jmp     .L29
.L44:
        pxor    %xmm0, %xmm0
        cvtsi2sd    %r11d, %xmm0
        divsd   .LC2(%rip), %xmm0
        movslq  %ebx, %rcx
        imulq   $19832, %rcx, %rcx
        cvttsd2si   %xmm0, %eax
        movl    %eax, 19824(%rsi,%rcx)
        addl    $1, %ebx
.L28:
        cmpl    %edi, %ebx
        jge     .L45
        movl    $0, %r10d
        movl    $0, %r11d
        jmp     .L33
.L45:
        movl    $0, %r8d
        jmp     .L34
.
..L35:
        addl    $1, %edx
.L37:
        cmpl    %edi, %edx
        jge     .L46
        movslq  %r8d, %rax
        imulq   $19832, %rax, %rax
        addq    %rsi, %rax
        movslq  %edx, %rcx
        imulq   $19832, %rcx, %rcx
        movl    19824(%rsi,%rcx), %ebx
        cmpl    %ebx, 19824(%rax)
        jge     .L35
        movl    19828(%rax), %ebx
        leal    1(%rbx), %ecx
        movl    %ecx, 19828(%rax)
        jmp     .L35
.L46:
        addl    $1, %r8d
.L34:
        cmpl    %edi, %r8d
        jge     .L47
        movl    $0, %edx
        jmp     .L37
.L47:
        popq    %rbx
        .cfi_def_cfa_offset 8
        ret
        .cfi_endproc
.LFE46:
        .size   calculate_momentum, .-
calculate_momentum
        .section    .rodata.str1.1
.LC3:
        .string "₩nrank %d stock
no. %d₩n"
.LC4:
        .string "date : profit₩n"
.LC5:
        .string "%d : %lf₩n"
.LC6:
        .string "₩nbuy_date's profit: %lf₩n"
        .section
        .rodata.str1.8,"aMS",@progbits,1
        .align 8
.LC7:
        .string "₩nrank%d max profit: %lf,
date : %d₩n₩n"
        .align 8
.LC8:
        .string "momentum of purchasing
date that satisfies minimum"
        .section    .rodata.str1.1
.LC9:
        .string "profit %d : %d₩n"
.LC10:
        .string "₩nfinished finding
stocks₩n"
        .text
        .globl  calculate_profit
        .type   calculate_profit, @function
```

그림 61 op5(optimization 5의 결과 코드) assembly

```
calculate_momentum:
.LFB46:
        .cfi_startproc
        pushq   %r12
        .cfi_def_cfa_offset 16
        .cfi_offset 12, -16
        pushq   %rbp
        .cfi_def_cfa_offset 24
        .cfi_offset 6, -24
        pushq   %rbx
        .cfi_def_cfa_offset 32
        .cfi_offset 3, -32
        movl    $0, %ebx
        jmp     .L25
.L26:
        movslq  %ebx, %rax
        imulq   $19832, %rax, %rax
        addq    %rsi, %rax
        movslq  %ecx, %r11
        movl    3968(%rax,%r11,4), %r9d
        leal    12(%rcx), %r10d
        movslq  %r10d, %r10
        subl    3968(%rax,%r10,4), %r9d
        movl    %r9d, 7932(%rax,%r11,4)
        addl    $1, %ecx
.L27:
        leal    -12(%r8), %eax
        cmpl    %ecx, %eax
        jg      .L26
        addl    $1, %ebx
.L25:
        cmpl    %edi, %ebx
        jge     .L43
        movl    $0, %ecx
        jmp     .L27
.L43:
        movl    $0, %ebx
        jmp     .L28
.L31:
        leal    (%r9,%r10), %eax
        movslq  %eax, %r12
        addl    7932(%rcx,%r12,4), %r11d
        addl    $1, %eax
        cltq
        addl    7932(%rcx,%rax,4), %ebp
        addl    $2, %r10d
.L29:
        cmpl    $11, %r10d
        jle     .L31
.L30:
        addl    $1, %r9d
.L33:
        cmpl    %r8d, %r9d
        jge     .L44
        movslq  %ebx, %rcx
        imulq   $19832, %rcx, %rcx
        addq    %rsi, %rcx
        movslq  %r9d, %rax
        cmpl    %edx, 4(%rcx,%rax,4)
        jne     .L30
        movl    $0, %r10d
        jmp     .L29
.L44:
        addl    %ebp, %r11d
        pxor    %xmm0, %xmm0
        cvtsi2sd    %r11d, %xmm0
        divsd   .LC2(%rip), %xmm0
        movslq  %ebx, %rcx
        imulq   $19832, %rcx, %rcx
        cvttsd2si   %xmm0, %eax
        movl    %eax, 19824(%rsi,%rcx)
        addl    $1, %ebx
```

```
.L28:
        cmpl    %edi, %ebx
        jge     .L45
        movl    $0, %r9d
        movl    $0, %ebp
        movl    $0, %r11d
        jmp     .L33
.L45:
        movl    $0, %r8d
        jmp     .L34
.L35:
        addl    $1, %edx
.L37:
        cmpl    %edi, %edx
        jge     .L46
        movslq  %r8d, %rax
        imulq   $19832, %rax, %rax
        addq    %rsi, %rax
        movslq  %edx, %rcx
        imulq   $19832, %rcx, %rcx
        movl    19824(%rsi,%rcx), %ebx
        cmpl    %ebx, 19824(%rax)
        jge     .L35
        movl    19828(%rax), %ebx
        leal    1(%rbx), %ecx
        movl    %ecx, 19828(%rax)
        jmp     .L35
.L46:
        addl    $1, %r8d
.L34:
        cmpl    %edi, %r8d
        jge     .L47
        movl    $0, %edx
        jmp     .L37
.L47:
        popq    %rbx
        .cfi_def_cfa_offset 24
        popq    %rbp
        .cfi_def_cfa_offset 16
        popq    %r12
        .cfi_def_cfa_offset 8
        ret
        .cfi_endproc
.LFE46:
        .size   calculate_momentum, .-
calculate_momentum
        .section    .rodata.str1.1
.LC3:
        .string "₩nrank %d stock no. %d₩n"
.LC4:
        .string "date : profit₩n"
.LC5:
        .string "%d : %lf₩n"
.LC6:
        .string "₩nbuy_date's profit: %lf₩n"
        .section    .rodata.str1.8,"aMS",@progbits,1
        .align 8
.LC7:
        .string "₩nrank%d max profit: %lf,
date : %d₩n₩n"
        .align 8
.LC8:
        .string "momentum of purchasing date that
satisfies minimum"
        .section    .rodata.str1.1
.LC9:
        .string "profit %d : %d₩n"
.LC10:
        .string "₩nfinished finding stocks₩n"
        .text
        .globl  calculate_profit
        .type   calculate_profit, @function
```

그림 62 op6(optimization 6의 결과 코드) assembly

op5와 op6의 assembly code 모두 %rsp를 이용하여 stack에 접근하는 부분은 보이지 않았기 때문에 register spill이 발생한 것은 아닌 것을 알 수 있다. 그러나 c코드에서는 2x2 loop unrolling이 일어나도록 코드를 작성했지만 assembly code를 비교해보면 그림 52, 그림 53에서 볼 수 있듯이 total1을 위한 코드와 total0 + total1을 계산하는 부분을 제외하면 코드에서 다른 점을 발견할 수 없다. 추가적으로 작성한 3x3 loop unrolling code에서도 같았다. 수업에서 loop unrolling은 하드웨어 레벨에서 일어나는 일이라고 배웠는데 이 때문에 코드에서는 차이를 보이지 않는 것으로 추정된다.[2]

assembly code를 확인하면서 calle saved registers에 대해서 함수의 주요 부분 실행 전 pushq, popq를 이용하여 stack에 값을 저장하고 다시 불러오는 과정을 확인할 수 있었던 것이 특이사항이다.

4) 다음 단계 optimization 내용

profile 결과도 input 파일의 용량을 키웠음에도 최소 단위에 이르러 더 이상 관찰할 수 없고 assembly code를 통해서도 profile의 결과를 알기 어렵기 때문에 더 이상 optimization을 진행하고 결과를 확인하는 것이 어렵다고 판단하여 optimization 6까지만 진행하기로 한다.

4. 최종 결론

최종적으로 6번의 optimization을 진행하였다. op1부터 op3까지는 loop내에서 중복적으로 call되는 고정 값을 외부에서 전달해주거나 loop 바깥에서 정의함으로써 불필요한 function call을 줄였다. op4에서는 값이 변경되지 않음에도 loop 내부에 정의된 변수를 loop 바깥으로 꺼내거나 값을 함수 외부에서 인자로 전달함으로써 loop inefficiency를 줄였다. op5에서는 프로그램을 실행하는데 불필요한 loop를 제거하여 실행시간을 줄였고 op6에서는 2x2 unrolling 함수를 구현했지만 실행시간이 'gprof'에서 표시하는 최소 단위 이하로 내려갔고 assembly 코드에서도 다른 점이 발견되지 않았기 때문에 소프트웨어 측면에서의 optimization은 마치기로 하였다. op4까지는 input file로 all_month_data.csv를 사용한 반면 op5부터는 all_stock_data.csv를 사용하고 이에 따라 struct STOCK의 크기도 달라진다.

5. 참조문헌

[1] “모멘텀 투자의 기본 원리.” Brunch, 29 Dec. 2016, brunch.co.kr/@boolio/3.

[2] Young Ik Eom. (2020). [Chap.5-2]Optimizing Program Performance[pdf].