## Find best func

De forskellige klasser bliver beregnet for sig, uden at kunne påvirke hinanden. Derfor for at finde det bedste skema der er blevet generet. Bliver alle de bedste skemaer for de enkelte generationer gået gennem, og der bliver udvalgt tre generationer. Den med den bedste 7. klasse, den bedste 8. klasse og dem bedste 9. klasse. Disse bliver nu sat sammen til deres egent individ, som bliver printet i print funktionen.

\begin{lstlisting}

void find\_best(individual chosen\_individuals[][NUMBER\_OF\_GENERATIONS], individual best\_of\_best[]){

printf("\n\n");

int i, k, j = 0;

int best\_sum1, best\_sum2, best\_sum3;

int best\_gen1, best\_gen2, best\_gen3;

int sum1, sum2, sum3;

int temp\_perfect1 = 0, temp\_perfect2 = 0, temp\_perfect3 = 0;

int perfect1 = 0, perfect2 = 0, perfect3 = 0;

for (i = 0; i < NUMBER\_OF\_GENERATIONS; i++){

/\* Getting the sum \*/

if ((chosen\_individuals[0][i].fitness != 1) && (chosen\_individuals[1][i].fitness != 1) && (chosen\_individuals[2][i].fitness != 1)){

sum1 = chosen\_individuals[0][i].fitness + chosen\_individuals[1][i].fitness + chosen\_individuals[2][i].fitness;

}

else {

sum1 = 0;

}

if ((chosen\_individuals[3][i].fitness != 1) && (chosen\_individuals[4][i].fitness != 1) && (chosen\_individuals[5][i].fitness != 1)){

sum2 = chosen\_individuals[3][i].fitness + chosen\_individuals[4][i].fitness + chosen\_individuals[5][i].fitness;

}

else {

sum2 = 0;

}

if ((chosen\_individuals[6][i].fitness != 1) && (chosen\_individuals[7][i].fitness != 1) && (chosen\_individuals[8][i].fitness != 1)){

sum3 = chosen\_individuals[6][i].fitness + chosen\_individuals[7][i].fitness + chosen\_individuals[8][i].fitness;

}

else {

sum3 = 0;

}

/\* Getting the lowest perfection grade \*/

temp\_perfect1 = 15;

temp\_perfect2 = 15;

temp\_perfect3 = 15;

if (temp\_perfect1 > chosen\_individuals[0][i].perfection){

temp\_perfect1 = chosen\_individuals[0][i].perfection;

}

if (temp\_perfect1 > chosen\_individuals[1][i].perfection){

temp\_perfect1 = chosen\_individuals[1][i].perfection;

}

if (temp\_perfect1 > chosen\_individuals[2][i].perfection){

temp\_perfect1 = chosen\_individuals[2][i].perfection;

}

if (temp\_perfect2 > chosen\_individuals[3][i].perfection){

temp\_perfect2 = chosen\_individuals[3][i].perfection;

}

if (temp\_perfect2 > chosen\_individuals[4][i].perfection){

temp\_perfect2 = chosen\_individuals[4][i].perfection;

}

if (temp\_perfect2 > chosen\_individuals[5][i].perfection){

temp\_perfect2 = chosen\_individuals[5][i].perfection;

}

if (temp\_perfect3 > chosen\_individuals[6][i].perfection){

temp\_perfect3 = chosen\_individuals[6][i].perfection;

}

if (temp\_perfect3 > chosen\_individuals[7][i].perfection){

temp\_perfect3 = chosen\_individuals[7][i].perfection;

}

if (temp\_perfect3 > chosen\_individuals[8][i].perfection){

temp\_perfect3 = chosen\_individuals[8][i].perfection;

}

/\* Saving the best \*/

if (perfect1 < temp\_perfect1){

best\_sum1 = sum1;

best\_gen1 = i;

perfect1 = temp\_perfect1;

printf(" 1 perfection: %d sum: %d \n", perfect1, best\_sum1);

}

else if (perfect1 == temp\_perfect1){

if (best\_sum1 < sum1){

best\_sum1 = sum1;

best\_gen1 = i;

perfect1 = temp\_perfect1;

printf(" 1 perfection: %d sum: %d \n", perfect1, best\_sum1);

}

}

if (perfect2 < temp\_perfect2){

best\_sum2 = sum2;

best\_gen2 = i;

perfect2 = temp\_perfect2;

printf(" 2 perfection: %d sum: %d \n", perfect2, best\_sum2);

}

else if (perfect2 == temp\_perfect2){

if (best\_sum2 < sum2){

best\_sum2 = sum2;

best\_gen2 = i;

perfect2 = temp\_perfect2;

printf(" 2 perfection: %d sum: %d \n", perfect2, best\_sum2);

}

}

if (perfect3 < temp\_perfect3){

best\_sum3 = sum3;

best\_gen3 = i;

perfect3 = temp\_perfect3;

printf(" 3 perfection: %d sum: %d \n", perfect3, best\_sum3);

}

else if (perfect3 == temp\_perfect3){

if (best\_sum3 < sum3){

best\_sum3 = sum3;

best\_gen3 = i;

perfect3 = temp\_perfect3;

printf(" 3 perfection: %d sum: %d \n", perfect3, best\_sum3);

}

}

}

best\_of\_best[0] = chosen\_individuals[0][best\_gen1];

best\_of\_best[1] = chosen\_individuals[1][best\_gen1];

best\_of\_best[2] = chosen\_individuals[2][best\_gen1];

best\_of\_best[3] = chosen\_individuals[3][best\_gen2];

best\_of\_best[4] = chosen\_individuals[4][best\_gen2];

best\_of\_best[5] = chosen\_individuals[5][best\_gen2];

best\_of\_best[6] = chosen\_individuals[6][best\_gen3];

best\_of\_best[7] = chosen\_individuals[7][best\_gen3];

best\_of\_best[8] = chosen\_individuals[8][best\_gen3];

for (j = 0; j < NUMBER\_OF\_CLASSES; j++){

best\_of\_best[j].best\_gena7 = best\_gen1;

best\_of\_best[j].best\_gena8 = best\_gen2;

best\_of\_best[j].best\_gena9 = best\_gen3;

}

printf("\n\n");

printf(" 1 perfection: %d sum: %d \n", perfect1, best\_sum1);

printf(" 2 perfection: %d sum: %d \n", perfect2, best\_sum2);

printf(" 3 perfection: %d sum: %d \n\n", perfect3, best\_sum3);

}

\end{lstlisting}

Denne funktion tager ’chosen\_individuals[][]’ og ’best\_of\_best[]’ som parametre. Det der sker at der først i funktionen er en for-løkke som går gennem antal generationer gange. Nu kommer der if-sætninger, hvis formål er at finde summen for parallelklasserne. Nu kommer programmet gennem yderligere if-sætninger, som finder den parallel klasse som har den laveste ’perfektion-grade’. Dette gør den da, vi leder efter den generation hvor en bestem klasse har den højeste perfektion-grade, og den højeste sum. Dog skal programmet først og fremmest, finde den klasse med den bedste perfektion grade.

Derfor kommer programmet nu gennem den sidste antal if-sætninger, som ser om perfektion-grade, er bedre end dem tidligere bedste. Hvis den nuværende generation er bedre end de forrige. Blive denne nu gemt.