```
#include <iostream>

using namespace std;

double player_stand(int ptot, int pace, int cards, int deck[]);
double player_hit(int ptot, int pace, int cards, int deck[]);
double player_double(int ptot, int pace, int deck[]);
double player_split(int ptot, int pace, int deck[]);
void dealer(int old_tot, int old_ace, int cards, double prob, int deck[], double dealer_prob[]);
double max2(double x, double y);

int numdeck = 6;
int upcard;

int main()
{
	int ch, i, j, deck[11], pc1, pc2;
	double dealer_prob[7], ev_stand, ev_hit, ev_double, ev_split, prob;
	for (i = 1; i <= 9; i++)
		deck[i] = 4 * numdeck;
	deck[10] = 16 * numdeck;
	deck[0] = 52 * numdeck;
	cerr << "1.  Dealer probabilities\n";
	cerr << "2.  Player hand\n";
	cerr << "3.  Whole game\n";
	cin >> ch;
	if (ch == 1)
	{
		for (i = 1; i <= 10; i++)
		{
			cout << "Up card = " << i << "\n";
			deck[i]--;
			deck[0]--;
			dealer(i, (i == 1 ? 1 : 0), 1, 1.0, deck, dealer_prob);
			for (j = 0; j <= 6; j++)
				cout << j << "\t" << dealer_prob[j] << "\n";
			deck[i]++;
			deck[0]++;
		}
	}
	else if (ch == 2)
	{
		cerr << "Player card 1:  ";
		cin >> pc1;
		cerr << "Player card 2:  ";
		cin >> pc2;
		cerr << "Dealer up card:  ";
		cin >> upcard;
		deck[pc1]--;
		deck[pc2]--;
		deck[upcard]--;
		deck[0] -= 3;
		ev_stand = player_stand(pc1 + pc2, (pc1 == 1 ? 1 : 0) + (pc2 == 1 ? 1 : 0), 2,
deck);
		ev_hit = player_hit(pc1 + pc2, (pc1 == 1 ? 1 : 0) + (pc2 == 1 ? 1 : 0), 2, deck);
		ev_double = player_double(pc1 + pc2, (pc1 == 1 ? 1 : 0) + (pc2 == 1 ? 1 : 0),
deck);
		if (pc1 == pc2)
			ev_split = player_split(pc1, (pc1 == 1 ? 1 : 0), deck);
		else
			ev_split = -1.0;
		cerr << "EV stand    =\t" << ev_stand << "\n";
		cerr << "EV hit      =\t" << ev_hit << "\n";
		cerr << "EV double   =\t" << ev_double << "\n";
		cerr << "EV split  =\t" << ev_split << "\n";
	}
	else if (ch == 3)
	{
		double prob = 1.0;
		double tot_prob = 0.0;
		double tot_ev = 0.0;
```

```
		double max_ev;
		int basic_strategy[11][11][11];
		for (pc1 = 1; pc1 <= 10; pc1++)
		{
			prob *= (double)deck[pc1] / (double)deck[0];
			deck[pc1]--;
			deck[0]--;
			for (pc2 = pc1; pc2 <= 10; pc2++)
			{
				prob *= (pc1 == pc2 ? 1 : 2) * (double)deck[pc2] / (double)deck[0];
				deck[pc2]--;
				deck[0]--;
				for (upcard = 1; upcard <= 10; upcard++)
				{
					prob *= (double)deck[upcard] / (double)deck[0];
					tot_prob += prob;
					deck[upcard]--;
					deck[0]--;
					ev_stand = player_stand(pc1 + pc2, (pc1 == 1 ? 1 : 0) +
(pc2 == 1 ? 1 : 0), 2, deck);
					ev_hit = player_hit(pc1 + pc2, (pc1 == 1 ? 1 : 0) + (pc2 ==
1 ? 1 : 0), 2, deck);
					ev_double = player_double(pc1 + pc2, (pc1 == 1 ? 1 : 0) +
(pc2 == 1 ? 1 : 0), deck);
					if (pc1 == pc2)
						ev_split = player_split(pc1, (pc1 == 1 ? 1 : 0),
deck);
					else
						ev_split = -1.0;
					max_ev = max2(max2(ev_hit, ev_stand), max2(ev_double,
ev_split));
					if (max_ev == ev_stand)
						basic_strategy[pc1][pc2][upcard] = 1;
					else if (max_ev == ev_hit)
						basic_strategy[pc1][pc2][upcard] = 2;
					else if (max_ev == ev_double)
						basic_strategy[pc1][pc2][upcard] = 3;
					else
						basic_strategy[pc1][pc2][upcard] = 4;
					tot_ev += prob * max_ev;
					printf("%i,%i,%i,%f,%f,%f,%f\n", pc1, pc2, upcard,
ev_stand, ev_hit, ev_double, ev_split);

					deck[0]++;
					deck[upcard]++;
					prob /= (double)deck[upcard] / (double)deck[0];
				}
				deck[0]++;
				deck[pc2]++;
				prob /= (pc1 == pc2 ? 1 : 2) * (double)deck[pc2] / (double)deck[0];
			}
			deck[0]++;
			deck[pc1]++;
			prob /= (double)deck[pc1] / (double)deck[0];
		}
		printf("Total probability   =\t%f\n", tot_prob);
		printf("Game Expected Value =\t%f\n", tot_ev);
		printf("Hard Totals\n");
		char codes[] = "XSHDP";
		for (pc1 = 2; pc1 <= 9; pc1++)
		{
			for (pc2 = pc1 + 1; pc2 <= 10; pc2++)
			{
				printf("%i+%i\t", pc1, pc2);
				for (upcard = 2; upcard <= 10; upcard++)
					printf("%c,", codes[basic_strategy[pc1][pc2][upcard]]);
				printf("%c\n", codes[basic_strategy[pc1][pc2][1]]);
			}
		}
		printf("Soft Totals\n");
		for (pc2 = 2; pc2 <= 10; pc2++)
```

```
                {
                        printf("%i+%i\t", 1, pc2);
                        for (upcard = 2; upcard <= 10; upcard++)
                                printf("%c,", codes[basic_strategy[1][pc2][upcard]]);
                        printf("%c\n", codes[basic_strategy[1][pc2][1]]);
                }
                printf("Pairs\n");
                for (pc2 = 1; pc2 <= 10; pc2++)
                {
                        printf("%i+%i\t", pc2, pc2);
                        for (upcard = 2; upcard <= 10; upcard++)
                                printf("%c,", codes[basic_strategy[pc2][pc2][upcard]]);
                        printf("%c\n", codes[basic_strategy[pc2][pc2][1]]);
                }
        }
}


double player_stand(int ptot, int pace, int cards, int deck[])
{
        double ev_stand, dealer_prob[7];
        if ((ptot == 11) && (pace == 1) && (cards == 2))
        {
                if ((upcard > 1) && (upcard < 10)) // winning BJ
                        return 1.5;
                else
                {
                        dealer(upcard, (upcard == 1 ? 1 : 0), 1, 1.0, deck, dealer_prob);
                        return 1.5 * (dealer_prob[0] + dealer_prob[1] + dealer_prob[2] +
dealer_prob[3] + dealer_prob[4] + dealer_prob[6]);
                }
        }
        else if (ptot > 21) // player busts
                return -1.0;
        else
        {
                if ((ptot <= 11) && (pace > 0)) // soft
                        ptot += 10;
                dealer(upcard, (upcard == 1 ? 1 : 0), 1, 1.0, deck, dealer_prob);
                ev_stand = dealer_prob[6];
                ev_stand -= dealer_prob[5];
                for (int i = 0; i <= 4; i++)
                {
                        if (ptot > i + 17)
                                ev_stand += dealer_prob[i];
                        else if (ptot < i + 17)
                                ev_stand -= dealer_prob[i];
                }
        }
        return ev_stand;
}



double player_hit(int ptot, int pace, int cards, int deck[])
{
        int next_card;
        double prob,ev_stand,ev_hit,ev_hit_again;
        ev_hit = 0.0;
        for (next_card = 1; next_card <= 10; next_card++)
        {
                prob = (double)deck[next_card] / (double)deck[0];
                deck[next_card]--;
                deck[0]--;
                ev_stand = player_stand(ptot + next_card, pace + (next_card == 1 ? 1 : 0), cards +
1, deck);
                if (ptot + next_card < 17)
                        ev_hit_again = player_hit(ptot + next_card, pace + (next_card == 1 ? 1 :
0), cards + 1, deck);
                else
                        ev_hit_again = -1;
```

```
                ev_hit += prob * max2(ev_stand, ev_hit_again);
                deck[next_card]++;
                deck[0]++;
        }
        return ev_hit;
}

double player_double(int ptot, int pace, int deck[])
{
        int next_card;
        double prob, ev_double;
        ev_double = 0.0;
        if (ptot > 11)
                ev_double = -1;
        else
        {
                for (next_card = 1; next_card <= 10; next_card++)
                {
                        prob = (double)deck[next_card] / (double)deck[0];
                        deck[next_card]--;
                        deck[0]--;
                        ev_double += prob*player_stand(ptot + next_card, pace + (next_card == 1 ?
1 : 0), 3, deck);
                        deck[next_card]++;
                        deck[0]++;
                }
        }
        return 2*ev_double;
}


double player_split(int ptot, int pace, int deck[])
{
        int next_card;
        double prob, ev_split,ev_stand,ev_hit;
        ev_split = 0.0;
        for (next_card = 1; next_card <= 10; next_card++)
        {
                prob = (double)deck[next_card] / (double)deck[0];
                deck[next_card]--;
                deck[0]--;
                ev_stand = player_stand(ptot + next_card, pace + (next_card == 1 ? 1 : 0), 3,
deck);
                if (ptot == 1)
                        ev_hit = -1.0;
                else
                        ev_hit = player_hit(ptot + next_card, pace + (next_card == 1 ? 1 : 0), 2,
deck);
                ev_split += prob * max2(ev_stand, ev_hit);
                deck[next_card]++;
                deck[0]++;
        }
        return 2 * ev_split;
}


double max2(double x, double y)
{
        if (x > y)
                return x;
        else
                return y;
}



/*
Dealer Prob
0 = 17
1 = 18
2 = 19
```

```
3 = 20
4 = 21
5 = bj
6 = bust */


void dealer(int old_tot, int old_ace, int cards, double prob, int deck[], double dealer_prob[])
{
        int i,next_card,new_tot,new_ace;
        if (cards == 1)
        {
                for (i = 0; i <= 6; i++)
                        dealer_prob[i] = 0.0;
        }
        for (next_card = 1; next_card <= 10; next_card++)
        {
                prob *= (double)deck[next_card] / (double)deck[0];
                deck[next_card]--;
                deck[0]--;
                new_tot = old_tot + next_card;
                new_ace = old_ace + (next_card == 1 ? 1 : 0);
                if ((cards == 1) && (new_tot == 11) && (new_ace == 1)) // blackjack
                        dealer_prob[5] += prob;
                else if (new_tot > 21)
                        dealer_prob[6] += prob;
                else if (new_tot >= 17)
                        dealer_prob[new_tot - 17] += prob;
                else if ((new_tot >= 7) && (new_tot <= 11) && (new_ace > 0))
                        dealer_prob[new_tot - 7] += prob;
                else
                        dealer(new_tot, new_ace, cards + 1, prob, deck, dealer_prob);
                deck[0]++;
                deck[next_card]++;
                prob /= (double)deck[next_card] / (double)deck[0];
        }
}
```