```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>
#define MAX_LINES 100
#define MAX_LEN 100
typedef struct {
    char var[20];
    int is_constant;
    int value;
} Symbol;
Symbol symbol_table[100];
int symbol_count = 0;
void trim(char *str) {
    char *start = str;
    char *end;
    while (isspace((unsigned char)*start)) start++;
    if (*start == 0) { *str = 0; return; }
    end = start + strlen(start) - 1;
    while (end > start && isspace((unsigned char)*end)) end--;
    *(end + 1) = 0;
    memmove(str, start, end - start + 2);
}
void normalize(char *line, char *out) {
    int j = 0;
    for (int i = 0; line[i]; i++) {
        if (strchr("=+-*/;", line[i])) {
            out[j++] = ' ';
            out[j++] = line[i];
            out[j++] = ' ';
        } else {
            out[j++] = line[i];
        } }
    out[j] = '\0';
    trim(out); }
int is_number(char *str) {
    if (*str == '\0') return 0;
    int i = 0;
    if (str[0] == '-') i++;
    for (; str[i]; i++) {
        if (!isdigit(str[i])) return 0;   }
    return 1;}
int lookup(char *var) {
    for (int i = 0; i < symbol_count; i++) {
        if (strcmp(symbol_table[i].var, var) == 0)
            return i;
    }
    return -1;
}
void update_symbol(char *var, int is_const, int val) {
    int idx = lookup(var);
    if (idx == -1) {
        strcpy(symbol_table[symbol_count].var, var);
        symbol_table[symbol_count].is_constant = is_const;
        symbol_table[symbol_count].value = val;
        symbol_count++;
    } else {
        symbol_table[idx].is_constant = is_const;
        symbol_table[idx].value = val;
    } }
int evaluate(char *op1, char *op, char *op2, int *result) {
    int val1_known = 0, val2_known = 0, val1 = 0, val2 = 0;
    if (is_number(op1)) { val1 = atoi(op1); val1_known = 1; }
    else {
        int idx = lookup(op1);
        if (idx != -1 && symbol_table[idx].is_constant) {
            val1 = symbol_table[idx].value;
            val1_known = 1;
        }  }
    if (is_number(op2)) { val2 = atoi(op2); val2_known = 1; }
    else {
        int idx = lookup(op2);
        if (idx != -1 && symbol_table[idx].is_constant) {
            val2 = symbol_table[idx].value;
            val2_known = 1;
        } }
    if (val1_known && val2_known) {
        if (strcmp(op, "+") == 0) *result = val1 + val2;
        else if (strcmp(op, "-") == 0) *result = val1 - val2;
        else if (strcmp(op, "*") == 0) *result = val1 * val2;
        else if (strcmp(op, "/") == 0 && val2 != 0) *result = val1 / val2;
        else return 0;
        return 1; }
    return 0; }
void process_line(char *line, char *optimized_line) {
    char norm[MAX_LEN];
    normalize(line, norm);
    char var[20], eq[5], op1[20], op[5], op2[20], semi[5];
    int matched = sscanf(norm, "%s %s %s %s %s %s", var, eq, op1, op, op2, semi);
    if (matched >= 3 && strcmp(eq, "=") == 0) {
        if (matched == 4) {  // form: x = y ;
            if (is_number(op1)) {
                update_symbol(var, 1, atoi(op1));
                sprintf(optimized_line, "%s = %s;", var, op1);
            } else {
                int idx = lookup(op1);
                if (idx != -1 && symbol_table[idx].is_constant) {
                    update_symbol(var, 1, symbol_table[idx].value);
                    sprintf(optimized_line, "%s = %d;", var, symbol_table[idx].value);
                } else {
                    update_symbol(var, 0, 0);
                    sprintf(optimized_line, "%s = %s;", var, op1);
                } }  } else if (matched >= 6) {  // form: x = y + z ;
            int result;
            if (evaluate(op1, op, op2, &result)) {
                update_symbol(var, 1, result);
                sprintf(optimized_line, "%s = %d;", var, result);
            } else{
                char left[20], right[20];
                int idx1 = lookup(op1);
                if (idx1 != -1 && symbol_table[idx1].is_constant)
                    sprintf(left, "%d", symbol_table[idx1].value);
                else strcpy(left, op1);
                int idx2 = lookup(op2);
                if (idx2 != -1 && symbol_table[idx2].is_constant)
                    sprintf(right, "%d", symbol_table[idx2].value);
                else strcpy(right, op2);
                sprintf(optimized_line, "%s = %s %s %s;", var, left, op, right);
                if (evaluate(left, op, right, &result)) {
                    update_symbol(var, 1, result);
                    sprintf(optimized_line, "%s = %d;", var, result);
                } else {
                    update_symbol(var, 0, 0);
                }    }   } else {
            strcpy(optimized_line, line);
        } } else {
        strcpy(optimized_line, line);  }}
int main() {
    int n;
    char lines[MAX_LINES][MAX_LEN];
    char optimized[MAX_LINES][MAX_LEN];
    printf("Enter number of lines: ");
    scanf("%d", &n);
    getchar();
    printf("Enter code lines:\n");
    for (int i = 0; i < n; i++) {
        fgets(lines[i], MAX_LEN, stdin);
        trim(lines[i]);
        if (strncmp(lines[i], "int ", 4) == 0) {
            memmove(lines[i], lines[i] + 4, strlen(lines[i]) - 3);
    }
        else if (strncmp(lines[i], "float ", 6) == 0) {
            memmove(lines[i], lines[i] + 6, strlen(lines[i]) - 5);
        }
        else if (strncmp(lines[i], "char ", 5) == 0) {
            memmove(lines[i], lines[i] + 5, strlen(lines[i]) - 4);
        } }
    printf("\nOptimized Code with Constant Propagation:\n");
    for (int i = 0; i < n; i++) {
        process_line(lines[i], optimized[i]);
        printf("%s\n", optimized[i]);
    }
    return 0; }
```