TASK 1

/\* task1.c \*/

#include <stdio.h>

#define print\_var(x) printf(#x " is %d", x )

const int y = 6;

int main() {

    int x = 5;

    print\_var(x);

    print\_var(5);

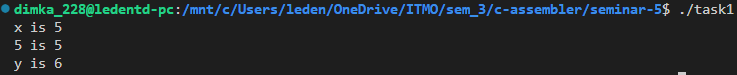
    print\_var(y);

    return 0;

}

Изображение выглядит как текст

Автоматически созданное описание



TASK 2

#include <stdint.h>

#include <stdlib.h>

#include <stdio.h>

#include <inttypes.h>

void int64\_t\_print(int64\_t i) {

    printf("%" PRId64 " ", i);

}

void double\_print(double d) {

    printf("%lf ", d);

}

struct list\_int64\_t\* create\_int64\_t\_node(int64\_t x);

struct list\_double\* create\_double\_node(double x);

#define \_print(type, list) list\_##type##\_print(list)

#define \_push(type, list, x) list\_##type##\_push(list, x)

#define DEFINE\_LIST(type)\

    struct list\_##type {\

        type value;\

        struct list\_##type\* next;\

    };                     \

    void list\_##type##\_print(struct list\_##type\* list) {\

        while (list != 0) {\

            type##\_print(list->value);\

            list = list->next;\

        }\

    }\

    void list\_##type##\_push(struct list\_##type\* list, type x) { \

        while (list->next != NULL) {  \

            list = list->next;\

        }                    \

        list->next = create\_##type##\_node(x);                     \

    }

DEFINE\_LIST(int64\_t)

DEFINE\_LIST(double)

struct list\_int64\_t\* create\_int64\_t\_node(int64\_t x) {

    struct list\_int64\_t\* node = malloc(sizeof(struct list\_int64\_t));

    node->value = x;

    return node;

}

struct list\_double\* create\_double\_node(double x) {

    struct list\_double\* node = malloc(sizeof(struct list\_double));

    node->value = x;

    return node;

}

int main() {

    struct list\_int64\_t\* list\_int = create\_int64\_t\_node(0);

    \_push(int64\_t, list\_int, 1);

    \_push(int64\_t, list\_int, 2);

    \_push(int64\_t, list\_int, 3);

    \_print(int64\_t, list\_int);

    printf("\n##############\n");

    struct list\_double\* list\_double = create\_double\_node(0);

    \_push(double, list\_double, 1.1);

    \_push(double, list\_double, 2.22);

    \_push(double, list\_double, 3.333);

    \_push(double, list\_double, 4.4444);

    \_print(double, list\_double);

}

Изображение выглядит как текст

Автоматически созданное описание

TASK 3

#include <stdint.h>

#include <stdlib.h>

#include <stdio.h>

#include <inttypes.h>

void int64\_t\_print(int64\_t i) {

    printf("%" PRId64 " ", i);

}

void double\_print(double d) {

    printf("%lf ", d);

}

struct list\_int64\_t\* create\_int64\_t\_node(int64\_t x);

struct list\_double\* create\_double\_node(double x);

#define \_print(type, list) list\_##type##\_print(list)

#define \_push(type, list, x) list\_##type##\_push(list, x)

#define DEFINE\_LIST(type)\

  struct list\_##type {\

    type value;\

    struct list\_##type\* next;\

  };                     \

  void list\_##type##\_print(struct list\_##type\* list) {\

    while (list != 0) {\

        type##\_print(list->value);\

        list = list->next;\

    }\

  }\

    void list\_##type##\_push(struct list\_##type\* list, type x) { \

        while (list->next != NULL) {  \

            list = list->next;\

        }                    \

        list->next = create\_##type##\_node(x);                     \

    }

DEFINE\_LIST(int64\_t)

DEFINE\_LIST(double)

struct list\_int64\_t\* create\_int64\_t\_node(int64\_t x) {

    struct list\_int64\_t\* node = malloc(sizeof(struct list\_int64\_t));

    node->value = x;

    return node;

}

struct list\_double\* create\_double\_node(double x) {

    struct list\_double\* node = malloc(sizeof(struct list\_double));

    node->value = x;

    return node;

}

void error(const char \*s) {

    fprintf(stderr, "%s", s);

    abort();

}

#define list\_print(list)                                                   \

    \_Generic((list),                                                       \

            struct list\_int64\_t\* : list\_int64\_t\_print(list),                            \

            struct list\_double\* : list\_double\_print(list),                              \

            default : error("GG GA GA"))

#define list\_push(list, x) \

    \_Generic((list),                                                       \

            struct list\_int64\_t\* : list\_int64\_t\_push(list, x),                            \

            struct list\_double\* : list\_double\_push(list, x),                              \

            default : error("GG GA GA"))

int main() {

    struct list\_int64\_t\* list\_int = create\_int64\_t\_node(0);

    //\_push(int64\_t, list\_int, 1);

    //\_push(int64\_t, list\_int, 2);

    //\_push(int64\_t, list\_int, 3);

    //\_print(int64\_t, list\_int);

    list\_push(list\_int, 1);

    list\_push(list\_int, 2);

    list\_push(list\_int, 3);

    list\_print(list\_int);

    printf("\n##############\n");

    struct list\_double\* list\_double = create\_double\_node(0);

    list\_push(list\_double, 1.1);

    list\_push(list\_double, 2.22);

    list\_push(list\_double, 3.333);

    list\_push(list\_double, 4.4444);

    list\_print(list\_double);

}

Изображение выглядит как текст

Автоматически созданное описание

TASK 4

/\* printer.c \*/

#include <inttypes.h>

#include <malloc.h>

#include <stdio.h>

#include <string.h>

struct AST {

    enum AST\_type { AST\_BINOP, AST\_UNOP, AST\_LIT } type;

    union {

        struct binop {

            enum binop\_type { BIN\_PLUS, BIN\_MINUS, BIN\_MUL, BIN\_DIV} type;

            struct AST \*left, \*right;

        } as\_binop;

        struct unop {

            enum unop\_type { UN\_NEG } type;

            struct AST \*operand;

        } as\_unop;

        struct literal {

            int64\_t value;

        } as\_literal;

    };

};

/\* DSL \*/

static struct AST \*newnode(struct AST ast) {

    struct AST \*const node = malloc(sizeof(struct AST));

    \*node = ast;

    return node;

}

struct AST \_lit(int64\_t value) {

    return (struct AST){AST\_LIT, .as\_literal = {value}};

}

struct AST \*lit(int64\_t value) {

    return newnode(\_lit(value));

}

struct AST \_unop(enum unop\_type type, struct AST \*operand) {

    return (struct AST){AST\_UNOP, .as\_unop = {type, operand}};

}

struct AST \*unop(enum unop\_type type, struct AST \*operand) {

    return newnode(\_unop(type, operand));

}

struct AST \_binop(enum binop\_type type, struct AST \*left, struct AST \*right) {

    return (struct AST){AST\_BINOP, .as\_binop = {type, left, right}};

}

struct AST \*binop(enum binop\_type type, struct AST \*left, struct AST \*right) {

    return newnode(\_binop(type, left, right));

}

#define DECLARE\_BINOP(fun, code)                                               \

  struct AST \*fun(struct AST \*left, struct AST \*right) {                       \

    return binop(BIN\_##code, left, right);                                     \

  }

DECLARE\_BINOP(add, PLUS)

DECLARE\_BINOP(mul, MUL)

DECLARE\_BINOP(sub, MINUS)

DECLARE\_BINOP(mod, DIV)

#undef DECLARE\_BINOP

#define DECLARE\_UNOP(fun, code)                                                \

  struct AST \*fun(struct AST \*operand) {                                       \

    return unop(UN\_##code, operand);                                           \

  }

DECLARE\_UNOP(neg, NEG)

#undef DECLARE\_UNOP

/\* printer \*/

static const char \*BINOPS[] = {

        [BIN\_PLUS] = "+", [BIN\_MINUS] = "-", [BIN\_MUL] = "\*", [BIN\_DIV] = "/"};

static const char \*UNOPS[] = {[UN\_NEG] = "-"};

typedef void(printer)(FILE \*, struct AST \*);

void print(FILE \*f, struct AST \*ast);

void print\_binop(FILE \*f, struct AST \*ast) {

    fprintf(f, "(");

    print(f, ast->as\_binop.left);

    fprintf(f, ")");

    fprintf(f, "%s", BINOPS[ast->as\_binop.type]);

    fprintf(f, "(");

    print(f, ast->as\_binop.right);

    fprintf(f, ")");

}

void print\_unop(FILE \*f, struct AST \*ast) {

    fprintf(f, "(%s", UNOPS[ast->as\_unop.type]);

    print(f, ast->as\_unop.operand);

    fprintf(f, ")");

}

void print\_lit(FILE \*f, struct AST \*ast) {

    fprintf(f, "%" PRId64, ast->as\_literal.value);

}

static printer \*ast\_printers[] = {

        [AST\_BINOP] = print\_binop, [AST\_UNOP] = print\_unop, [AST\_LIT] = print\_lit};

void print(FILE \*f, struct AST \*ast) {

    if (ast)

        ast\_printers[ast->type](f, ast);

    else

        fprintf(f, "<NULL>");

}

int main() {

    struct AST \*left\_1 = lit(999);

    struct AST \*right\_2 = lit(728);

    struct AST \*ast = binop(BIN\_PLUS, left\_1, right\_2);

    print(stdout,ast);

    printf("\n");

    struct AST \*left = lit(2);

    struct AST \*right = lit(9);

    struct AST \*ast2 = binop(BIN\_MUL, left, right);

    struct AST \*left1 = lit(4);

    struct AST \*ast\_final = binop(BIN\_PLUS, left1, ast2);

    print(stdout, ast\_final);

    printf("\n");

    struct AST \*first = lit(3);

    struct AST \*sec = lit(5);

    struct AST \*sum = binop(BIN\_PLUS, first, sec);

    struct AST \*third = lit(9);

    struct AST \*fourth = lit(7);

    struct AST \*div = binop(BIN\_DIV, third, fourth);

    struct AST \*mul = binop(BIN\_MUL, sum, div);

    print(stdout,mul);

    return 0;

}

