Alexandria University
Faculty of Engineering
Computer and Systems Engineering Dept.
CS 321 – Programming Languages and Translators

Quafios Compiler Collection
Semantic Analysis and Code Generation

***Teamwork**:*

- *Mostafa Abd El-Aziz (65).*

Used Data Structures:

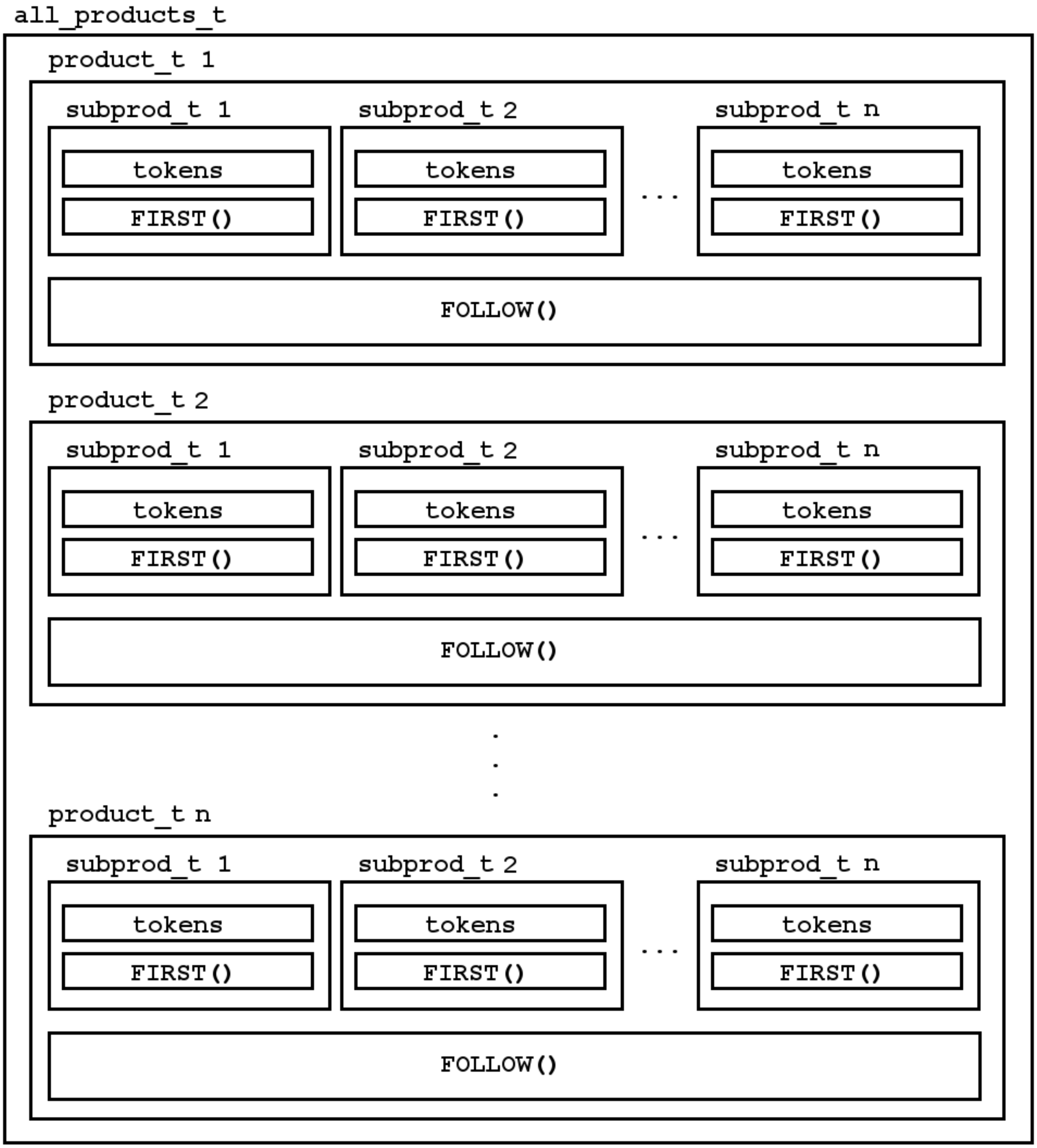
In this phase, I modified the parser generator to allow the generation of “semantic actions” which is processed by the parser in run-time. The parser generator takes context free grammar with semantic actions as input, and produces a C file that contains the code for the parser, along with semantic actions (each semantic block is coded as a single function) and an array of pointers to the functions.

- <u>**Representation of Production Rules**</u>:

```
/* if we have a rule A --> a b | c d (for example),
 * each of a, b, c, and d are called tokens, and
 * represented by token_t. 'a b' and 'c d' are called
 * sub-productions, and represented by subprod_t.
 * The whole production rule of A is represented by prod_t.
 * Every production rule has its own FOLLOW as a linked list
 * of token_t. Similarly, every sub-production has its own FIRST.
 */

/* tokens */
typedef struct token {
    struct token *next;
#define TOKEN_TYPE_SYMBOL      0 /* terminal */
#define TOKEN_TYPE_EPSILON     1
#define TOKEN_TYPE_UNION       2
#define TOKEN_TYPE_ASSIGN      3
#define TOKEN_TYPE_PRODUCT     4 /* nonterminal */
#define TOKEN_TYPE_DOLARSIGN   5
    int  type;
    char *data;
} token_t;

/* sub-production rules */
typedef struct subprod {
    struct subprod *next;
    token_t *tok_head; /* first token */
    token_t *tok_tail; /* last token */
    token_t *first; /* FIRST(subprod)*/
    int count;
} subprod_t;

/* production rules */
typedef struct product {
    struct product *next;
    char *name; /* name of the production rule */
    subprod_t *sub_head; /* sub production rules - head */
    subprod_t *sub_tail; /* sub production rules - tail */
    token_t *follow; /* FOLLOW(product)*/
    int firdone; /* first has been computed */
    int foldone; /* follow has been computed */
    int tmp; /* used by ptable construction */
} product_t;
```

```
/* list of all production rules in the program */
typedef struct {
    product_t *head;
    product_t *tail;
} all_products_t;
```

The following diagram illustrates the data structures used to represent production rules, and shows the hierarchy of the structures:

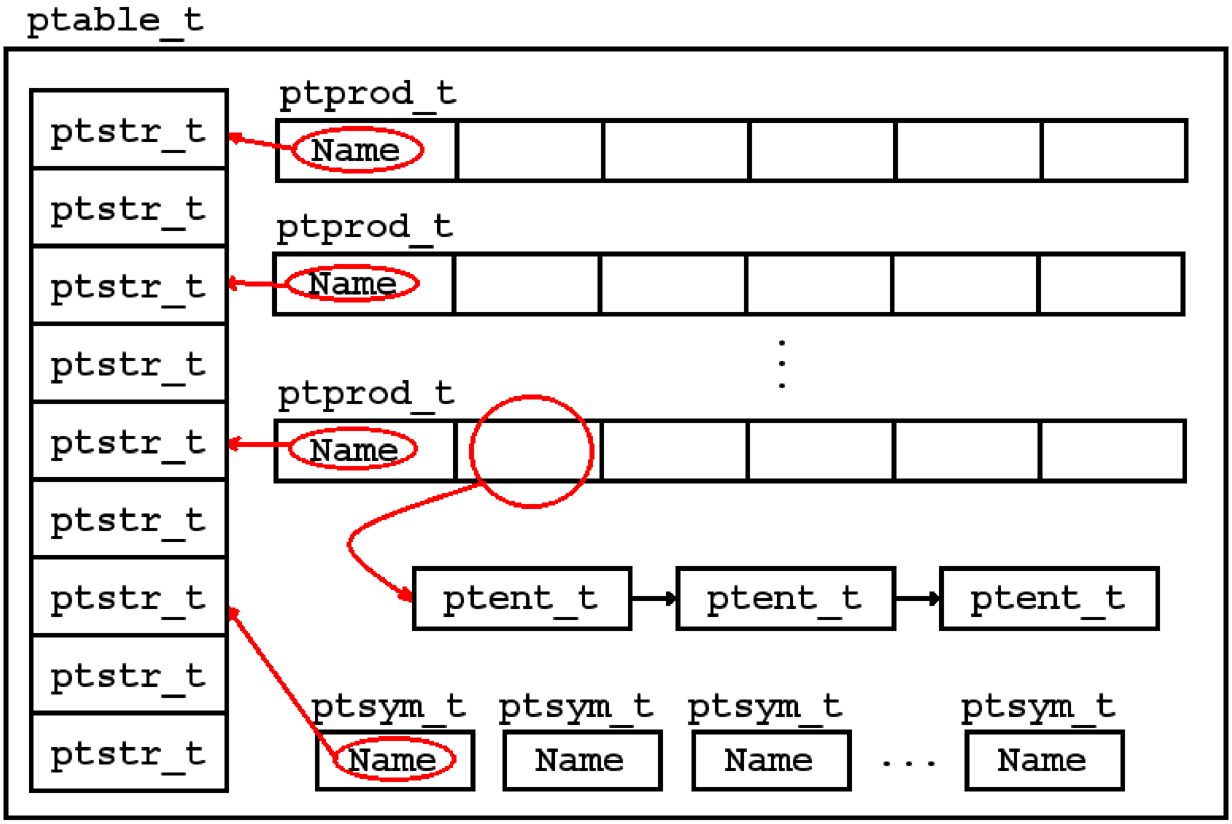


In any production: P → {action} X {action} Y {action} Z {action}; X, Y, and Z are encoded as ptent_t (parse table entry) elements. In this phase, I added code to parser generator so that {action} blocks are also encoded as ptent_t elements just like other tokens. In the case of {action} blocks, ptent_t structure will hold the index of the function (in semantic function pointer array, see above) that performs the actions specified in the block.

- **Representation of Parse table:**

```
/* Parse table is a matrix of M production rules x N symbols.
 * Every row of the table is represented by a ptprod_t element.
 * Every entry of the table is represented by a list of ptent_t elements.
 * The data structure representing parse table (ptable_t)
 * entirely contains all ptprod_t instances, all ptent_t
 * instances, and a list of terminal symbols of the language
 * (ptsym_t). All strings representing the symbols and production
 * names are stored in a string pool (ptstr_t).
 */

typedef struct ptsym {
    struct ptsym *next;
    int name_ptr; /* pointer to string holding the name in the pool */
    /* these used for comparisons, not stored in binary file */
    int index;
    char *name;
} ptsym_t;

typedef struct ptent {
    struct ptent *next;
#define PTENT_TYPE_TERMINATOR   0
#define PTENT_TYPE_PRODUCT      1
#define PTENT_TYPE_SYMBOL       2
#define PTENT_TYPE_EPSILON      3
    int type; /* type of the entry (pointer to production rule or symbol) */
    int ptr;  /* value of the pointer (index of the prod rule or symbol) */
} ptent_t;

typedef struct ptprod {
    /* row of the parse table */
    int name_ptr;  /* pointer to the string holding production name */
    int *subprods; /* array of entries of the row (N entries)
                      each entry corresponds to symbol
                      in the language */
    /* these are not stored in binary file */
    char *name;
} ptprod_t;

typedef struct ptstr {
    struct ptstr *next;
    char *str;
} ptstr_t;

typedef struct ptable {
    int symcount;  /* N-1 */
    int entcount;
    int prodcount; /* M */
    int chrcount;
    ptsym_t  *ptsyms;   /* symbols of the language */
    ptent_t  *ptents;   /* parts of production rules */
    ptprod_t *ptprods;  /* array of parse table rows */
    ptstr_t  *ptstrs;   /* pool of strings */
} ptable_t;
```

The following diagram illustrates the data structures used to represent parsing table, and shows the hierarchy of the structures:



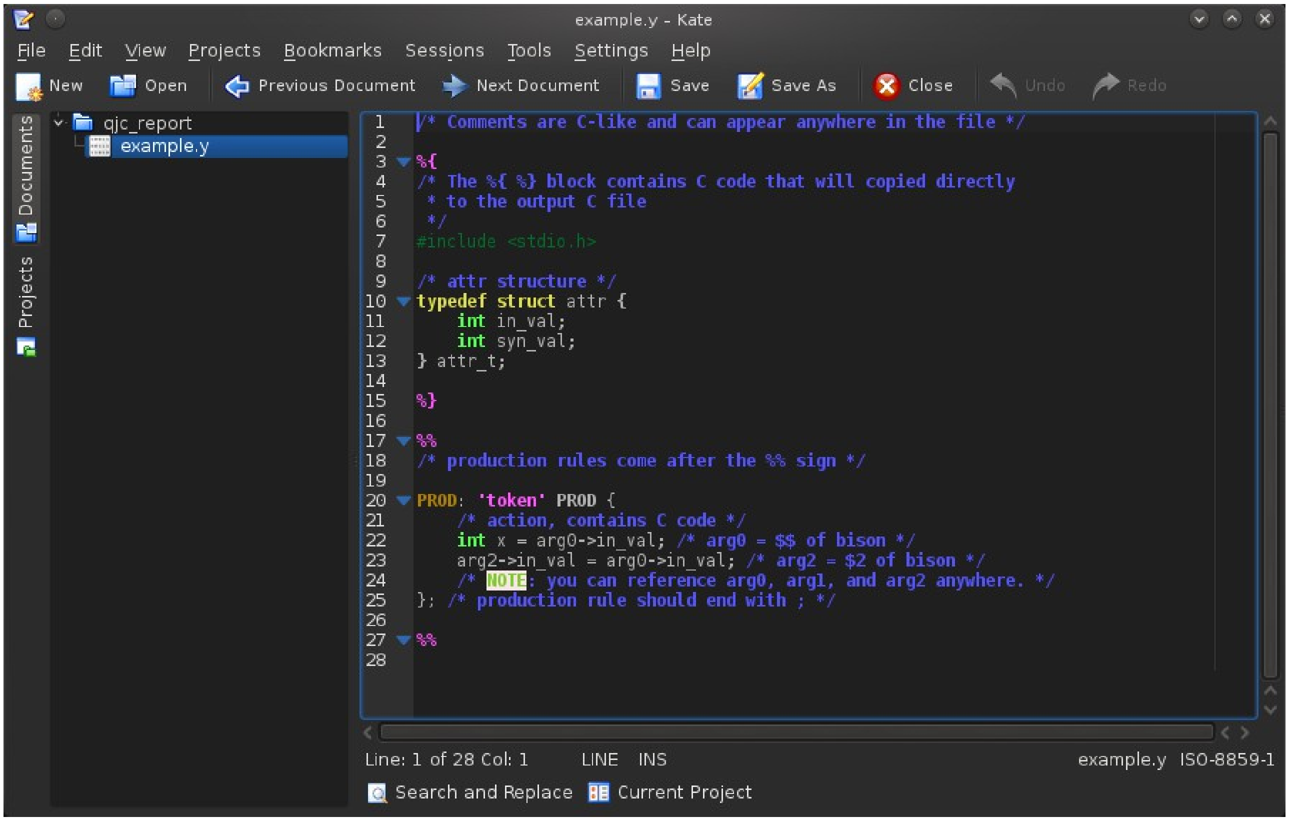
- **<u>Attribute structure:</u>**

Each semantic actions has a data structure that holds its attributes. The contents and size of the structure are defined by the language specification itself. Ever production has a linked list that holds the attribute structures of all tokens (and the root production), the data structure is called (attribute frame) and is allocated once production is removed from the simulation stack to be replaced with ptent_t elements.

Algorithms and Techniques:

- **Parsing of context free grammar and semantic actions:**

  The parser generator takes CFG as input and produces a C file that contains the parser, parse table, and semantic actions as output. The syntax of the input CFG is like bison syntax, and is illustrated below:



```
/* Comments are C-like and can appear anywhere in the file */

%{
/* The %{ %} block contains C code that will copied directly
 * to the output C file
 */
#include <stdio.h>

/* attr structure */
typedef struct attr {
    int in_val;
    int syn_val;
} attr_t;

%}

%%
/* production rules come after the %% sign */

PROD: 'token' PROD {
    /* action, contains C code */
    int x = arg0->in_val; /* arg0 = $$ of bison */
    arg2->in_val = arg0->in_val; /* arg2 = $2 of bison */
    /* NOTE: you can reference arg0, arg1, and arg2 anywhere. */
}; /* production rule should end with ; */

%%

```

- **Processing of semantic actions on run-time:**

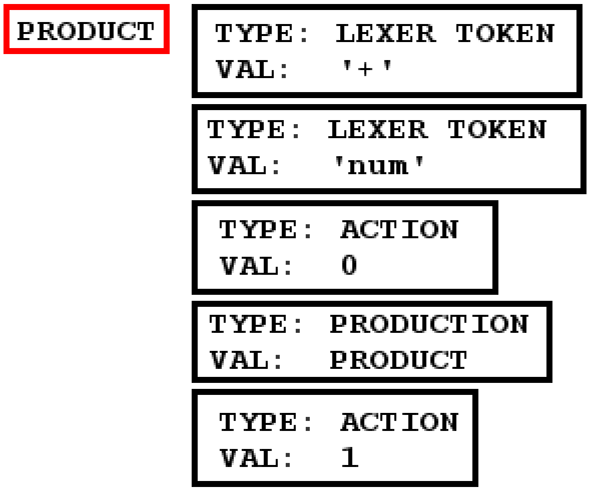
  The following example shows how semantic actions are compiled along with the generated parser and the generated parse table.

```
PRODUCT: '+' 'num' {
    /* calculate sum */
    arg3->in_val = arg0->in_val + atoi(curlexeme);
} PRODUCT {
    /* return result */
    arg0->syn_val = arg3->syn_val;
};
```

  In the output C file, two functions are generated for the two semantic actions above, and an array of pointers to those functions is created.

```
void _func0() {
 attr_ptr_t *frame = get attribute frame of current production;
 arg0 = get attribute structure 0 from the frame;
 arg1 = get attribute structure 1 from the frame;
 arg2 = get attribute structure 2 from the frame;
 arg3 = get attribute structure 3 from the frame;
 /* calculate sum */
 arg3->in_val = arg0->in_val + atoi(curlexeme);
}

void _func1() {
 attr_ptr_t *frame = get attribute frame of current production;
 arg0 = get attribute structure 0 from the frame;
 arg1 = get attribute structure 1 from the frame;
 arg2 = get attribute structure 2 from the frame;
 arg3 = get attribute structure 3 from the frame;
 /* return result */
 arg0->syn_val = arg3->syn_val;
}

void (*func_calls[2]) () = {
 _func0,
 _func1
};
```

The parse table will be generated as a byte array. The byte array is like a binary file that contains names of tokens, productions, ptent_t entries, etc... Please refer to the report of last phase to know the format of the parse table file. The ptent_t structures of PRODUCT production will be as follows:



PRODUCT

TYPE: LEXER TOKEN
VAL: '+'

TYPE: LEXER TOKEN
VAL: 'num'

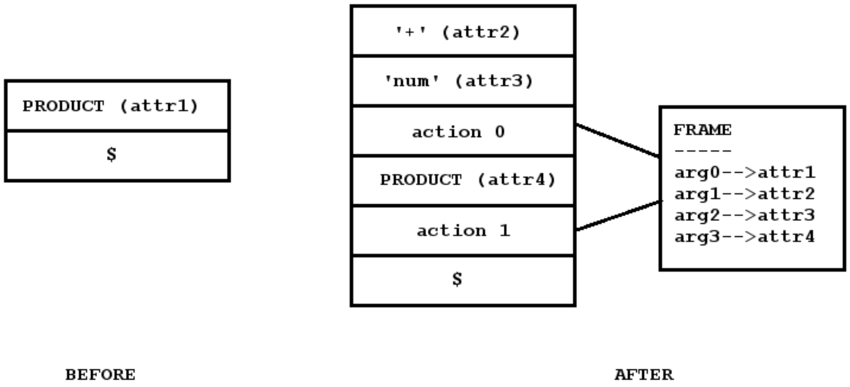
TYPE: ACTION
VAL: 0

TYPE: PRODUCTION
VAL: PRODUCT

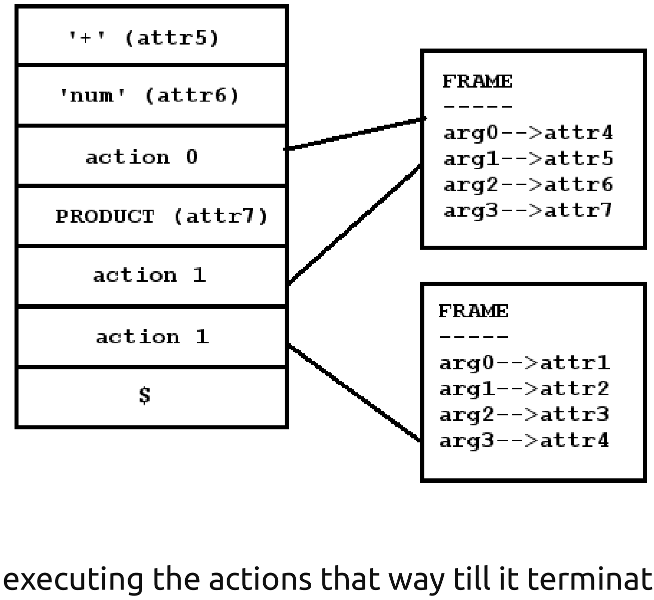
TYPE: ACTION
VAL: 1

When the parser begins, a “$” and PRODUCT are pushed into the stack. An attribute structure is allocated for the PRODUCT entry. When product is removed from the stack, it is replaced with the five pten_t structures specified above.

Each pten_t, when inserted into stack, is allocated an attribute structure (unless it is an action). Once PRODUCT is removed and replaced, an attribute frame is allocated with arg0 = attribute structure of the removed PRODUCT, and arg1 to arg3 are equal to the new attribute structures of the pten_t entries. The frame is accessible by all the semantic actions that derived from PRDOUCT when it was removed. The following diagram shows what happens to the stack:



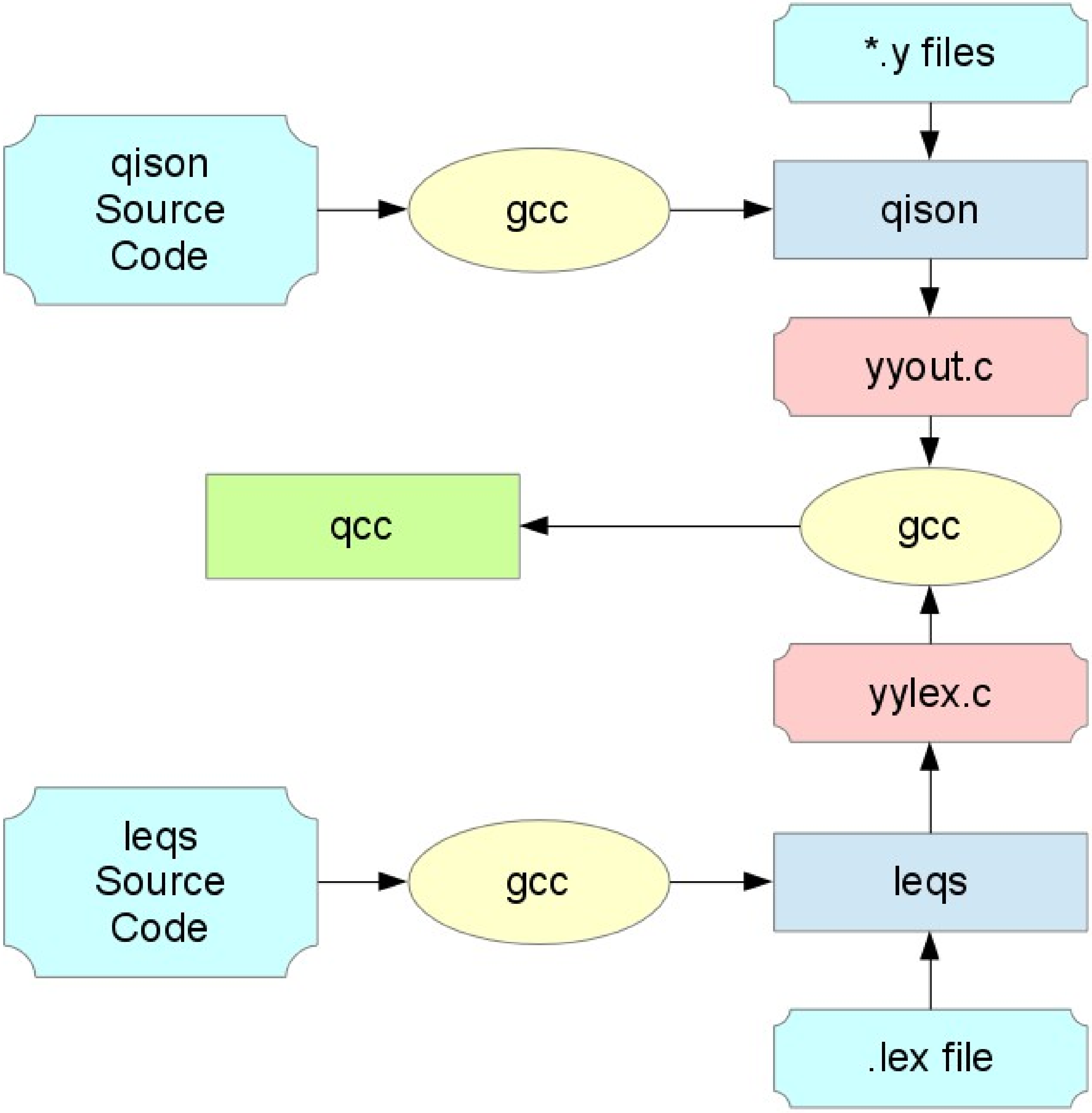
When '+' is matched, the '+' entry is removed from stack, but attr2 is not deallocated beacause it is still referenced by the frame that is associated with action 0 and action 1. The same happens when 'num' is matched. action 0 is then executed: the address of _func0 is fetched from the pointer array “func_calls” and a far call is performed. When PRODUCT is removed from stack, it is replaced with the same 5 tokens, but a new frame is allocated this time, and attr4 is inherited to the new frame as its arg0.



The parser keeps executing the actions that way till it terminates.

Makefile:

The following figure shows the stages of building a compiler using Quafios compiler compiler. The compiler compiler consists of a lexer generator (leqs) and a parser generator (qison).



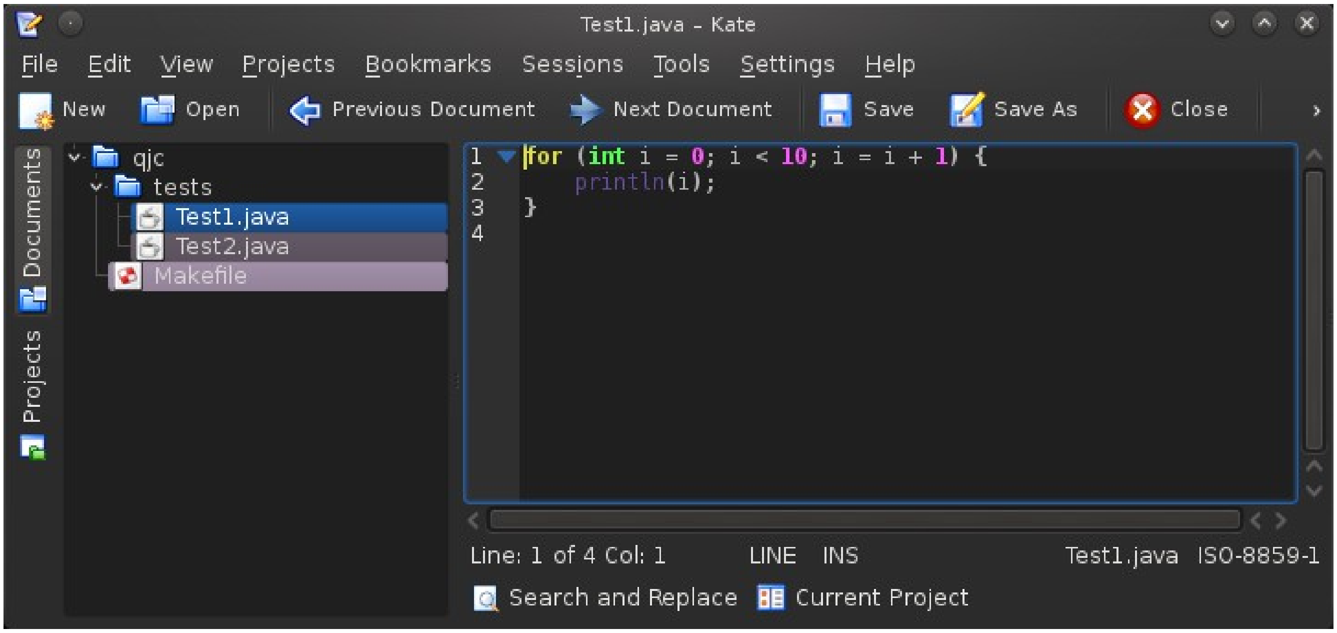
Quafios Java Compiler:

Using Quafios compiler generator, I managed to make a compiler for subset of Java programming language that supports the following:

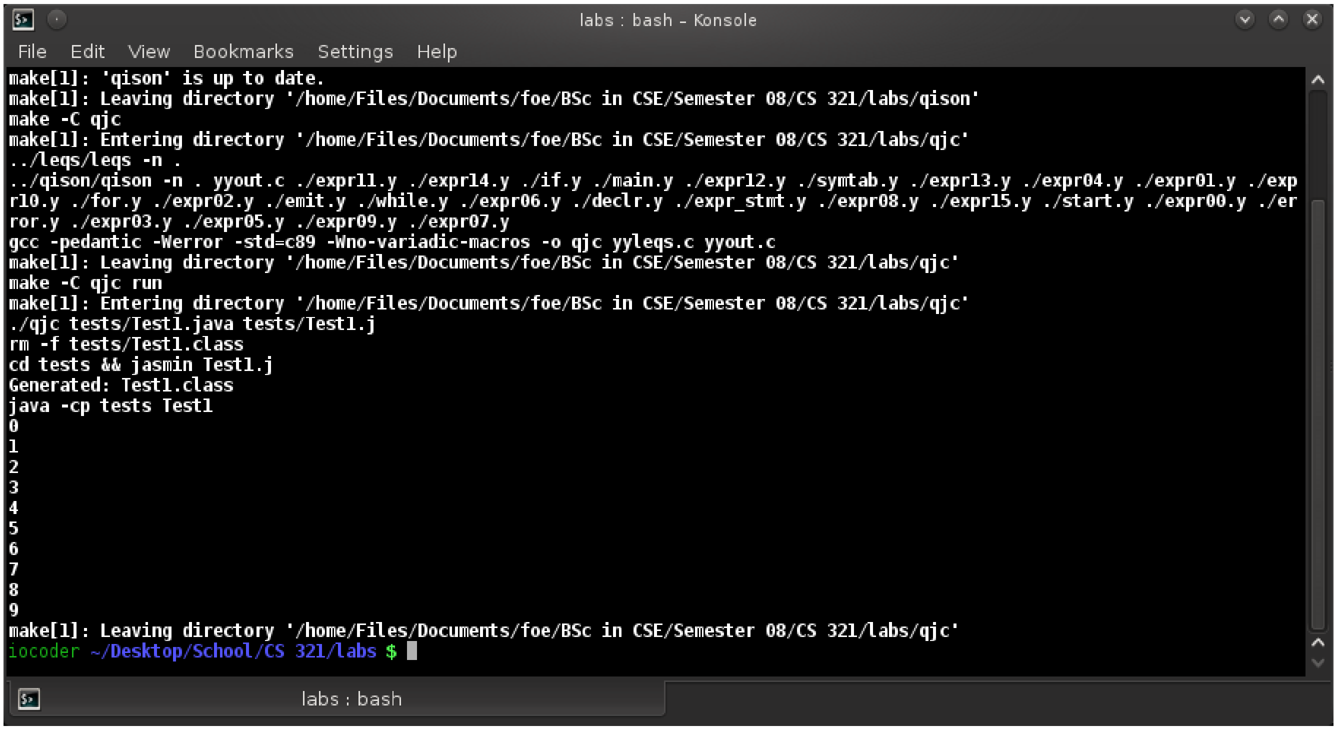
- Primitive Types (int, float, boolean)
- Arithmetic operations (+, -, *, /, %)
- Logical operations (&&, ||, !)
- Relational operations (>, >=, <, <=, ==, !=)
- If-else statements
- while loops
- Bitwise operations (&, |, ~) (**bonus**)
- boolean expressions (**bonus**)
- for loops (**bonus**)

The compiler maintains a symbol table to hold the names and types of variables in the source program. The compiler contains routines for emitting Java bytecode and routines for handling semantic errors and error reporting. Here are some code examples:

Example 1:

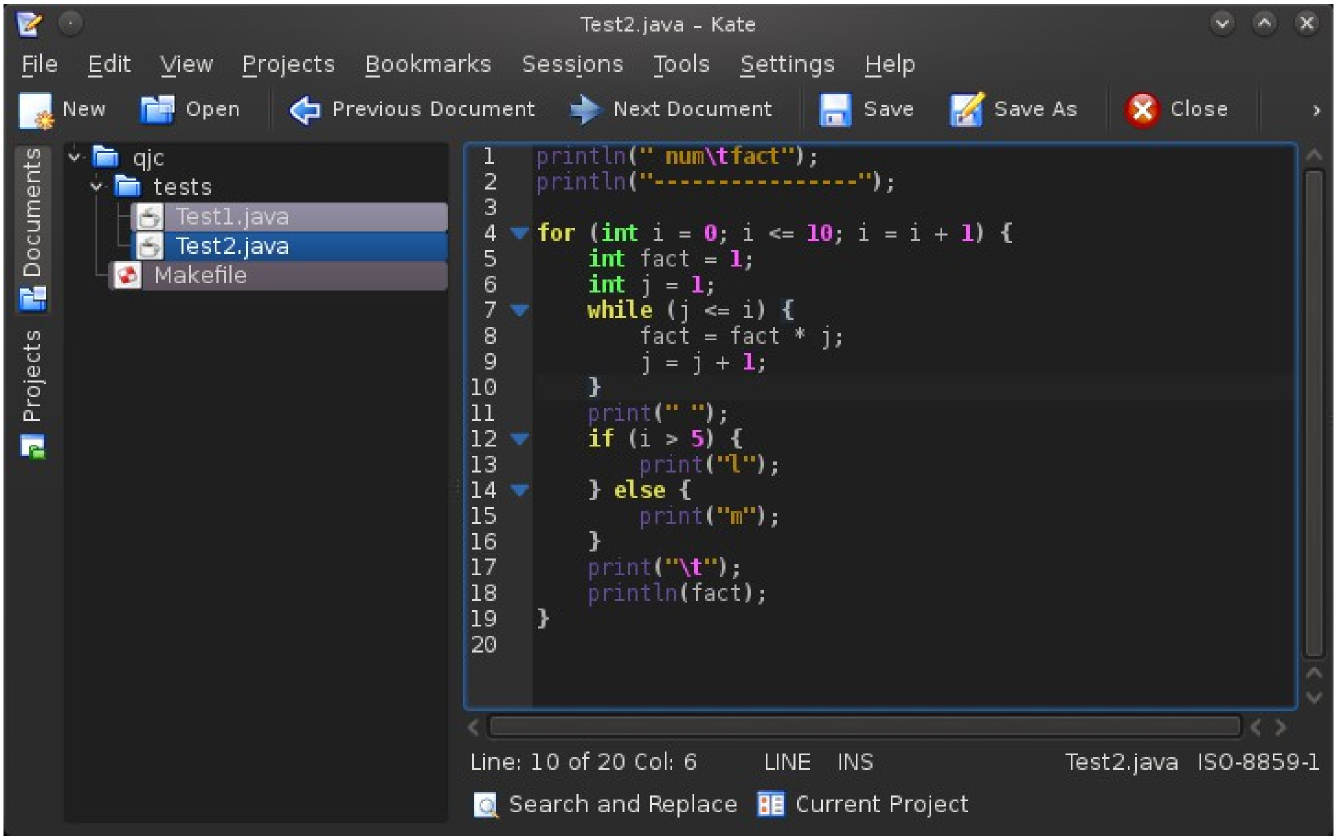


```
for (int i = 0; i < 10; i = i + 1) {
    println(i);
}
```

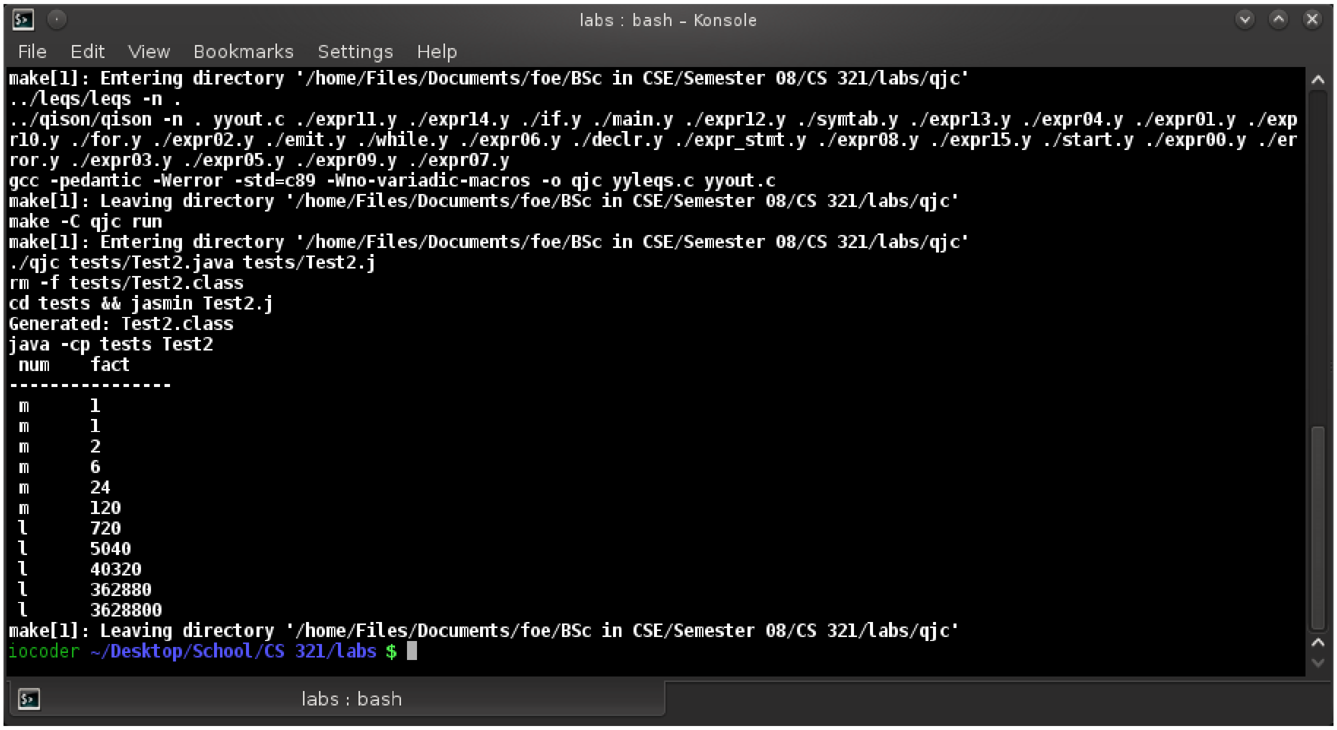


```
make[1]: 'qison' is up to date.
make[1]: Leaving directory '/home/Files/Documents/foe/BSc in CSE/Semester 08/CS 321/labs/qison'
make -C qjc
make[1]: Entering directory '/home/Files/Documents/foe/BSc in CSE/Semester 08/CS 321/labs/qjc'
../leqs/leqs -n .
../qison/qison -n . yyout.c ./expr11.y ./expr14.y ./if.y ./main.y ./expr12.y ./symtab.y ./expr13.y ./expr04.y ./expr01.y ./exp
r10.y ./for.y ./expr02.y ./emit.y ./while.y ./expr06.y ./declr.y ./expr_stmt.y ./expr08.y ./expr15.y ./start.y ./expr00.y ./er
ror.y ./expr03.y ./expr05.y ./expr09.y ./expr07.y
gcc -pedantic -Werror -std=c89 -Wno-variadic-macros -o qjc yyleqs.c yyout.c
make[1]: Leaving directory '/home/Files/Documents/foe/BSc in CSE/Semester 08/CS 321/labs/qjc'
make -C qjc run
make[1]: Entering directory '/home/Files/Documents/foe/BSc in CSE/Semester 08/CS 321/labs/qjc'
./qjc tests/Test1.java tests/Test1.j
rm -f tests/Test1.class
cd tests && jasmin Test1.j
Generated: Test1.class
java -cp tests Test1
0
1
2
3
4
5
6
7
8
9
make[1]: Leaving directory '/home/Files/Documents/foe/BSc in CSE/Semester 08/CS 321/labs/qjc'
iocoder ~/Desktop/School/CS 321/labs $ 
```

Example 2:



```
println(" num\tfact");
println("----------------");

for (int i = 0; i <= 10; i = i + 1) {
    int fact = 1;
    int j = 1;
    while (j <= i) {
        fact = fact * j;
        j = j + 1;
    }
    print(" ");
    if (i > 5) {
        print("l");
    } else {
        print("m");
    }
    print("\t");
    println(fact);
}

```



```
make[1]: Entering directory '/home/Files/Documents/foe/BSc in CSE/Semester 08/CS 321/labs/qjc'
../leqs/leqs -n .
../qison/qison -n . yyout.c ./expr11.y ./expr14.y ./if.y ./main.y ./expr12.y ./symtab.y ./expr13.y ./expr04.y ./expr01.y ./exp
r10.y ./for.y ./expr02.y ./emit.y ./while.y ./expr06.y ./declr.y ./expr_stmt.y ./expr08.y ./expr15.y ./start.y ./expr00.y ./er
ror.y ./expr03.y ./expr05.y ./expr09.y ./expr07.y
gcc -pedantic -Werror -std=c89 -Wno-variadic-macros -o qjc yyleqs.c yyout.c
make[1]: Leaving directory '/home/Files/Documents/foe/BSc in CSE/Semester 08/CS 321/labs/qjc'
make -C qjc run
make[1]: Entering directory '/home/Files/Documents/foe/BSc in CSE/Semester 08/CS 321/labs/qjc'
./qjc tests/Test2.java tests/Test2.j
rm -f tests/Test2.class
cd tests && jasmin Test2.j
Generated: Test2.class
java -cp tests Test2
 num    fact
----------------
 m      1
 m      1
 m      2
 m      6
 m      24
 m      120
 l      720
 l      5040
 l      40320
 l      362880
 l      3628800
make[1]: Leaving directory '/home/Files/Documents/foe/BSc in CSE/Semester 08/CS 321/labs/qjc'
iocoder ~/Desktop/School/CS 321/labs $ 
```

Quafios C Compiler (bonus):

Using Quafios compiler compiler, I managed to write a compiler for subset of C programming language that supports the following:

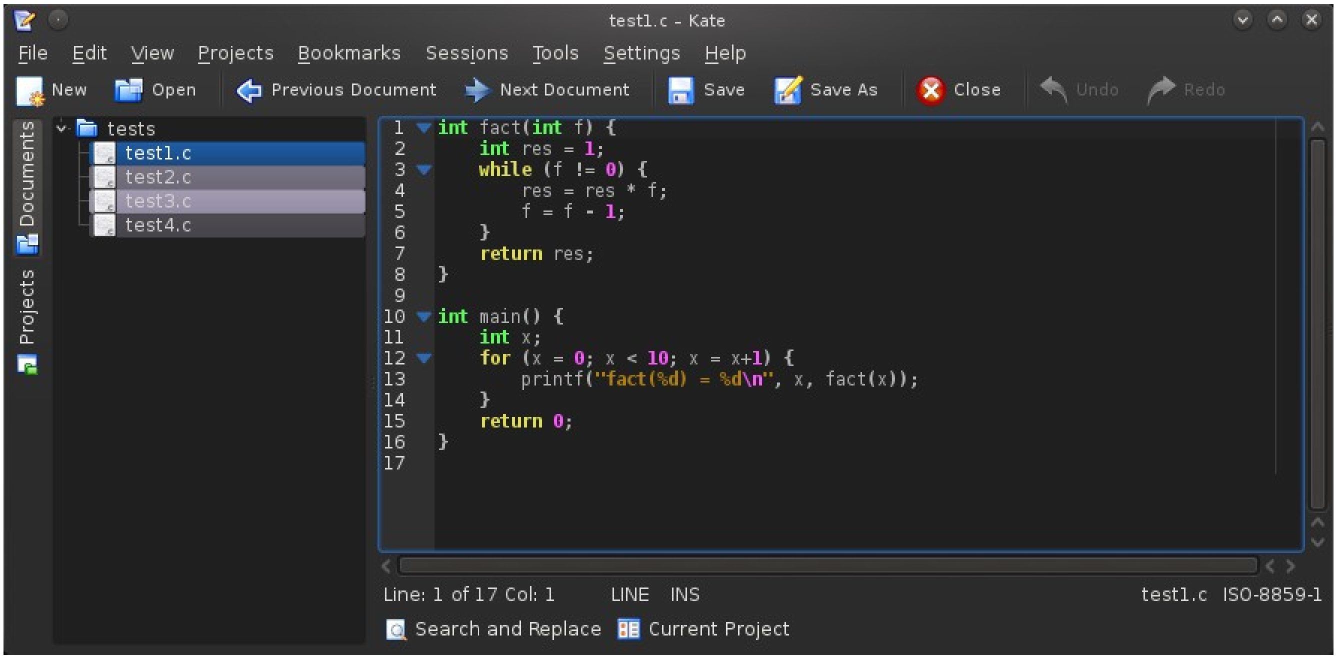
- Arithmetic operations (+, -, *, /, %)
- Logical operations (&&, ||, !)
- Relational operations (>, >=, <, <=, ==, !=)
- If-else statements
- while loops
- Bitwise operations (&, |, ~)
- boolean expressions
- for loops
- **Variable scope**
- **Global variables**
- **Function (and parameters) definition**
- **Function calls**
- **Parameter passing (by value and by reference)**
- **C arrays**

The C compiler generates i386 assembly code for GNU/Linux operating system. The generated assembly code is assembled by GNU assembler (as), then linked with glibc and other necessary libraries using GNU linker/loader (ld). The generated binary is a 32-bit binary ELF file that can run on any GNU/Linux system on x86.

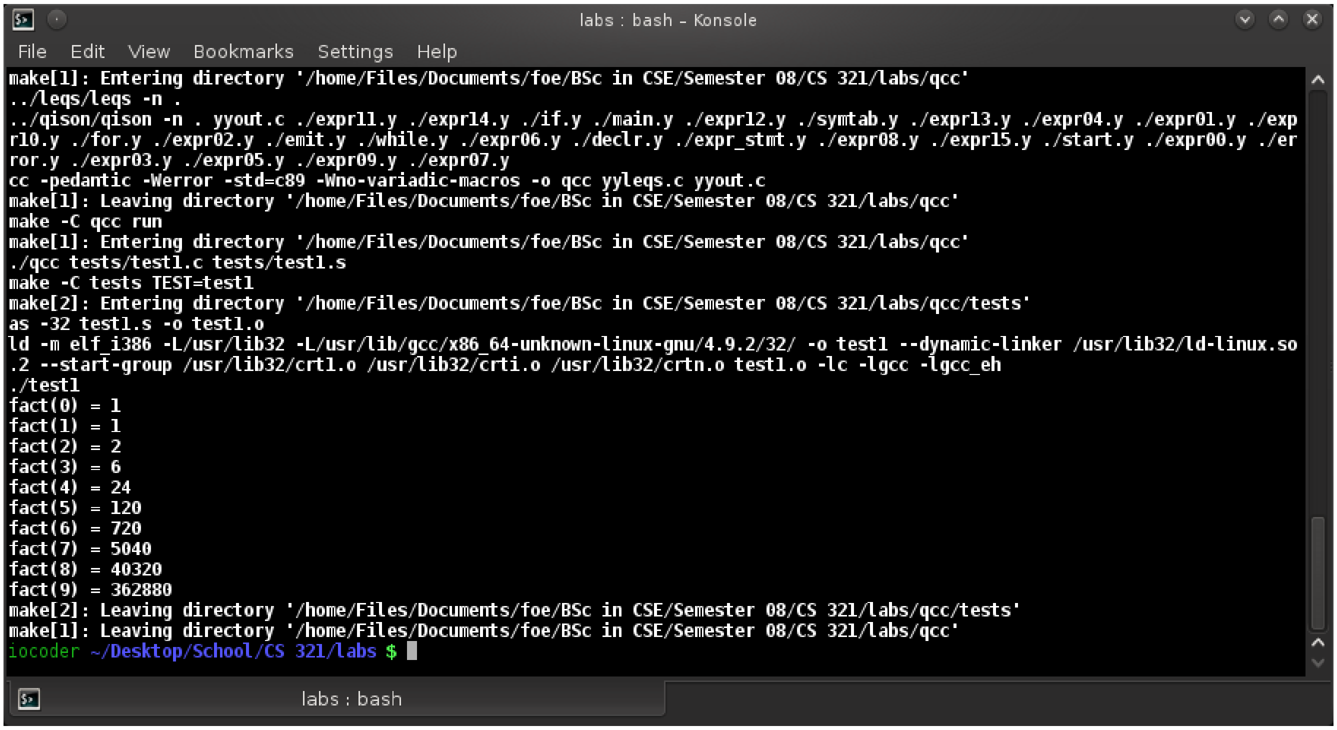
References used: ANSI standard specification for C language (1989/1990).

Following are screen-shots for 5 example programs compiled by qcc.

Example 1:

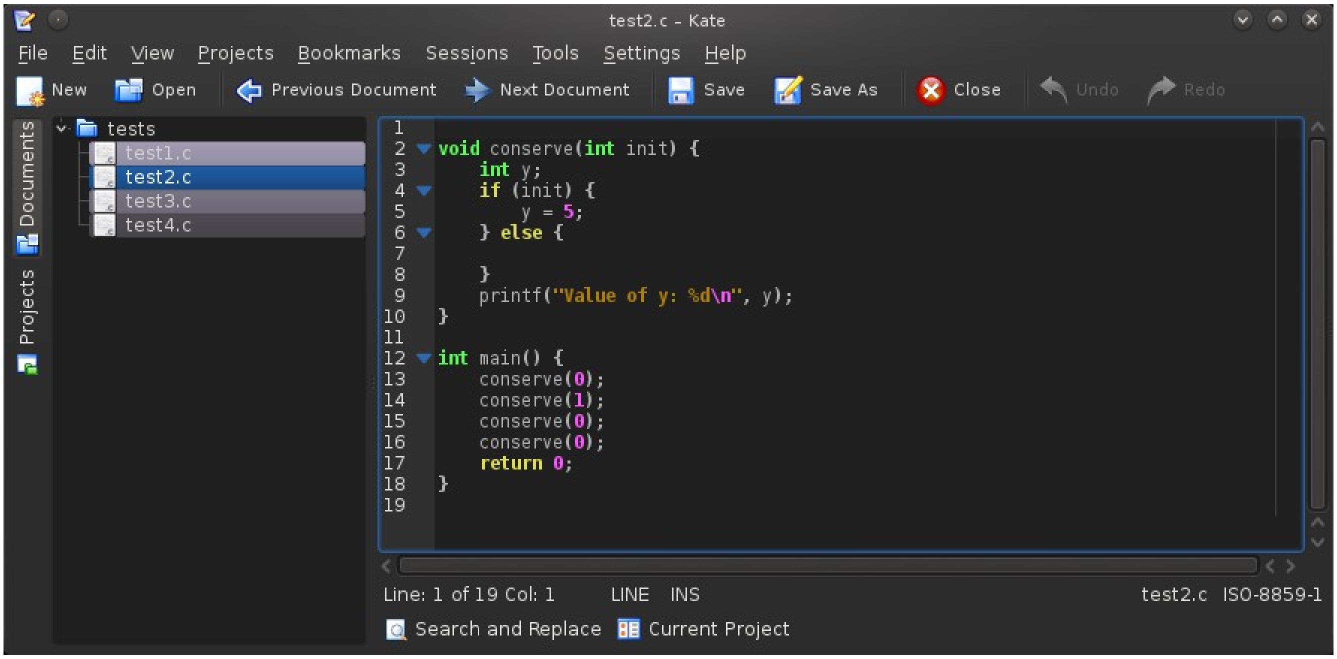

test1.c – Kate
File Edit View Projects Bookmarks Sessions Tools Settings Help
New Open Previous Document Next Document Save Save As Close Undo Redo
Documents
Projects
tests
test1.c
test2.c
test3.c
test4.c
Line: 1 of 17 Col: 1 LINE INS test1.c ISO-8859-1
Search and Replace Current Project

```c
int fact(int f) {
    int res = 1;
    while (f != 0) {
        res = res * f;
        f = f - 1;
    }
    return res;
}

int main() {
    int x;
    for (x = 0; x < 10; x = x+1) {
        printf("fact(%d) = %d\n", x, fact(x));
    }
    return 0;
}
```

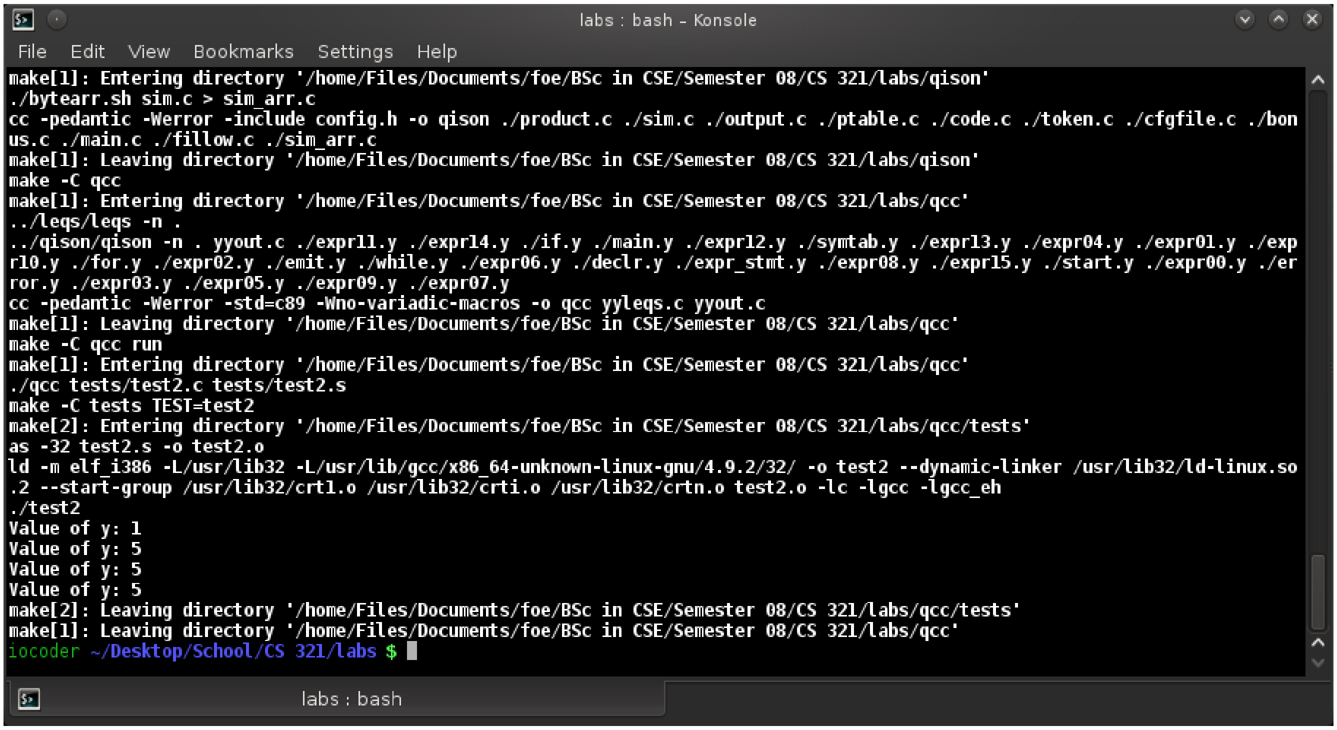


```
make[1]: Entering directory '/home/Files/Documents/foe/BSc in CSE/Semester 08/CS 321/labs/qcc'
../leqs/leqs -n .
../qison/qison -n . yyout.c ./expr11.y ./expr14.y ./if.y ./main.y ./expr12.y ./symtab.y ./expr13.y ./expr04.y ./expr01.y ./exp
r10.y ./for.y ./expr02.y ./emit.y ./while.y ./expr06.y ./declr.y ./expr_stmt.y ./expr08.y ./expr15.y ./start.y ./expr00.y ./er
ror.y ./expr03.y ./expr05.y ./expr09.y ./expr07.y
cc -pedantic -Werror -std=c89 -Wno-variadic-macros -o qcc yyleqs.c yyout.c
make[1]: Leaving directory '/home/Files/Documents/foe/BSc in CSE/Semester 08/CS 321/labs/qcc'
make -C qcc run
make[1]: Entering directory '/home/Files/Documents/foe/BSc in CSE/Semester 08/CS 321/labs/qcc'
./qcc tests/test1.c tests/test1.s
make -C tests TEST=test1
make[2]: Entering directory '/home/Files/Documents/foe/BSc in CSE/Semester 08/CS 321/labs/qcc/tests'
as -32 test1.s -o test1.o
ld -m elf_i386 -L/usr/lib32 -L/usr/lib/gcc/x86_64-unknown-linux-gnu/4.9.2/32/ -o test1 --dynamic-linker /usr/lib32/ld-linux.so
.2 --start-group /usr/lib32/crt1.o /usr/lib32/crti.o /usr/lib32/crtn.o test1.o -lc -lgcc -lgcc_eh
./test1
fact(0) = 1
fact(1) = 1
fact(2) = 2
fact(3) = 6
fact(4) = 24
fact(5) = 120
fact(6) = 720
fact(7) = 5040
fact(8) = 40320
fact(9) = 362880
make[2]: Leaving directory '/home/Files/Documents/foe/BSc in CSE/Semester 08/CS 321/labs/qcc/tests'
make[1]: Leaving directory '/home/Files/Documents/foe/BSc in CSE/Semester 08/CS 321/labs/qcc'
iocoder ~/Desktop/School/CS 321/labs $ 
```

Example 2:

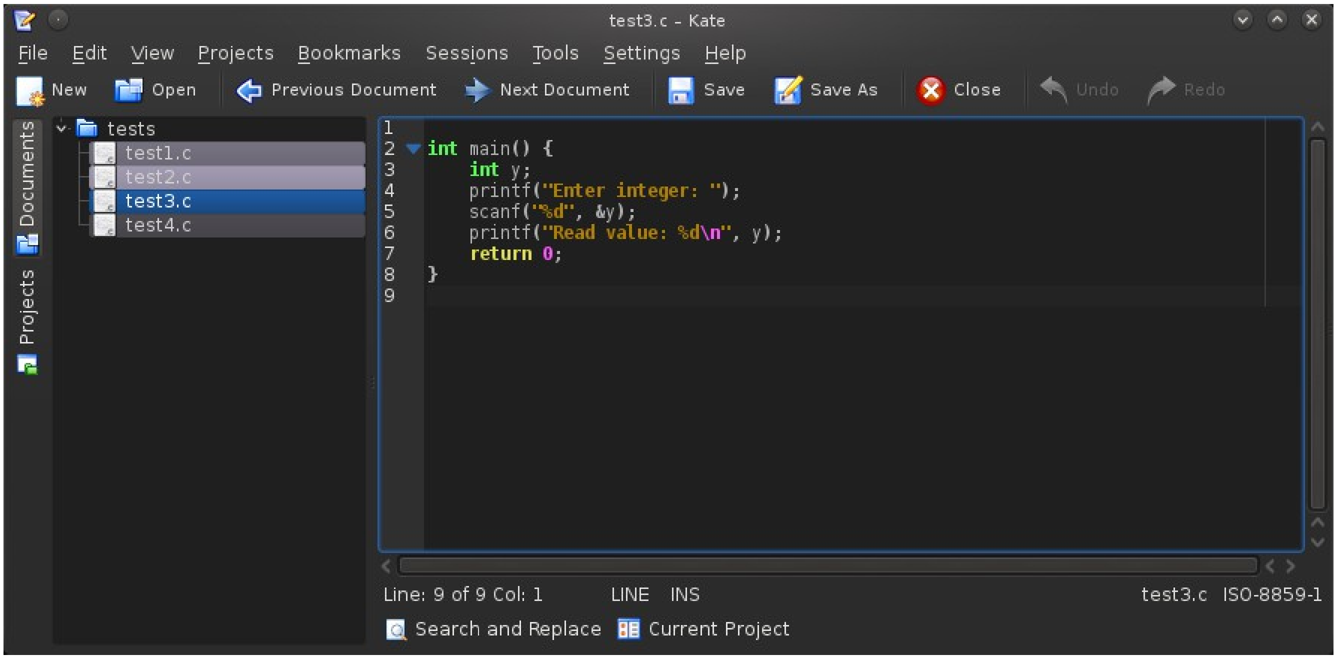


```

void conserve(int init) {
    int y;
    if (init) {
        y = 5;
    } else {

    }
    printf("Value of y: %d\n", y);
}

int main() {
    conserve(0);
    conserve(1);
    conserve(0);
    conserve(0);
    return 0;
}

```

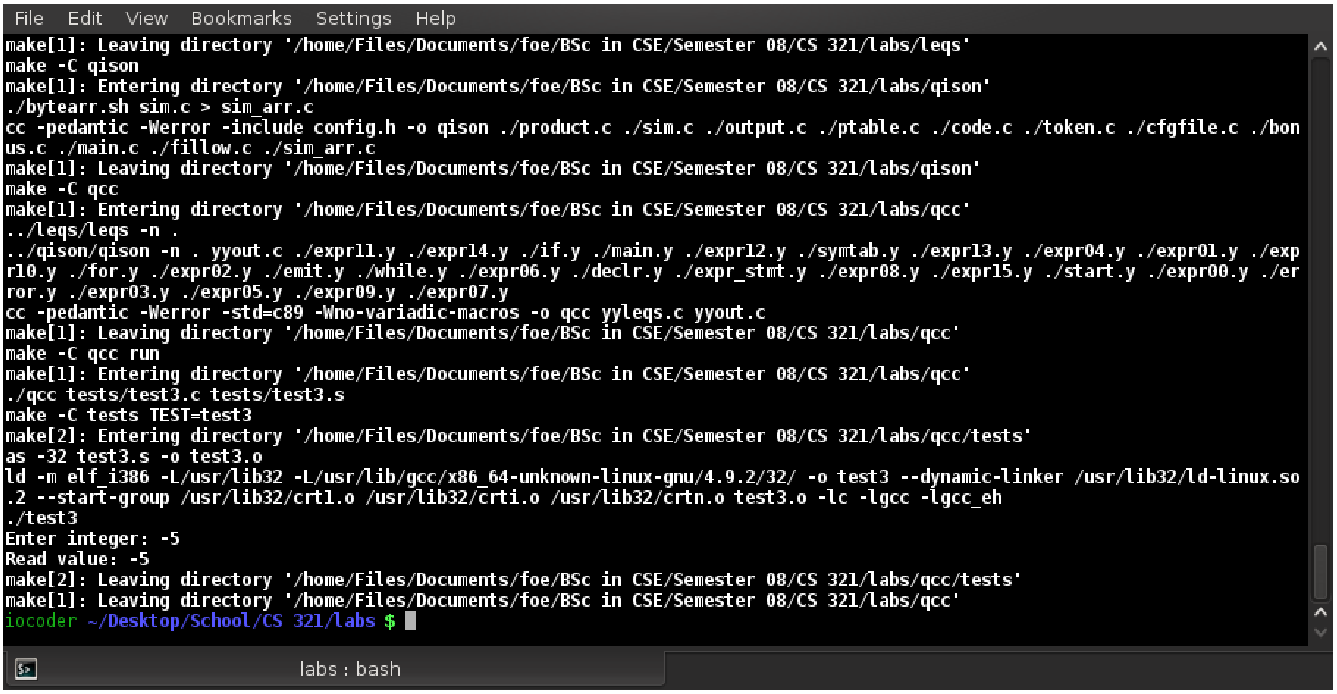


```
make[1]: Entering directory '/home/Files/Documents/foe/BSc in CSE/Semester 08/CS 321/labs/qison'
./bytearr.sh sim.c > sim_arr.c
cc -pedantic -Werror -include config.h -o qison ./product.c ./sim.c ./output.c ./ptable.c ./code.c ./token.c ./cfgfile.c ./bon
us.c ./main.c ./fillow.c ./sim_arr.c
make[1]: Leaving directory '/home/Files/Documents/foe/BSc in CSE/Semester 08/CS 321/labs/qison'
make -C qcc
make[1]: Entering directory '/home/Files/Documents/foe/BSc in CSE/Semester 08/CS 321/labs/qcc'
../leqs/leqs -n .
../qison/qison -n . yyout.c ./expr11.y ./expr14.y ./if.y ./main.y ./expr12.y ./symtab.y ./expr13.y ./expr04.y ./expr01.y ./exp
r10.y ./for.y ./expr02.y ./emit.y ./while.y ./expr06.y ./declr.y ./expr_stmt.y ./expr08.y ./expr15.y ./start.y ./expr00.y ./er
ror.y ./expr03.y ./expr05.y ./expr09.y ./expr07.y
cc -pedantic -Werror -std=c89 -Wno-variadic-macros -o qcc yyleqs.c yyout.c
make[1]: Leaving directory '/home/Files/Documents/foe/BSc in CSE/Semester 08/CS 321/labs/qcc'
make -C qcc run
make[1]: Entering directory '/home/Files/Documents/foe/BSc in CSE/Semester 08/CS 321/labs/qcc'
./qcc tests/test2.c tests/test2.s
make -C tests TEST=test2
make[2]: Entering directory '/home/Files/Documents/foe/BSc in CSE/Semester 08/CS 321/labs/qcc/tests'
as -32 test2.s -o test2.o
ld -m elf_i386 -L/usr/lib32 -L/usr/lib/gcc/x86_64-unknown-linux-gnu/4.9.2/32/ -o test2 --dynamic-linker /usr/lib32/ld-linux.so
.2 --start-group /usr/lib32/crt1.o /usr/lib32/crti.o /usr/lib32/crtn.o test2.o -lc -lgcc -lgcc_eh
./test2
Value of y: 1
Value of y: 5
Value of y: 5
Value of y: 5
make[2]: Leaving directory '/home/Files/Documents/foe/BSc in CSE/Semester 08/CS 321/labs/qcc/tests'
make[1]: Leaving directory '/home/Files/Documents/foe/BSc in CSE/Semester 08/CS 321/labs/qcc'
iocoder ~/Desktop/School/CS 321/labs $ 
```

Example 3:

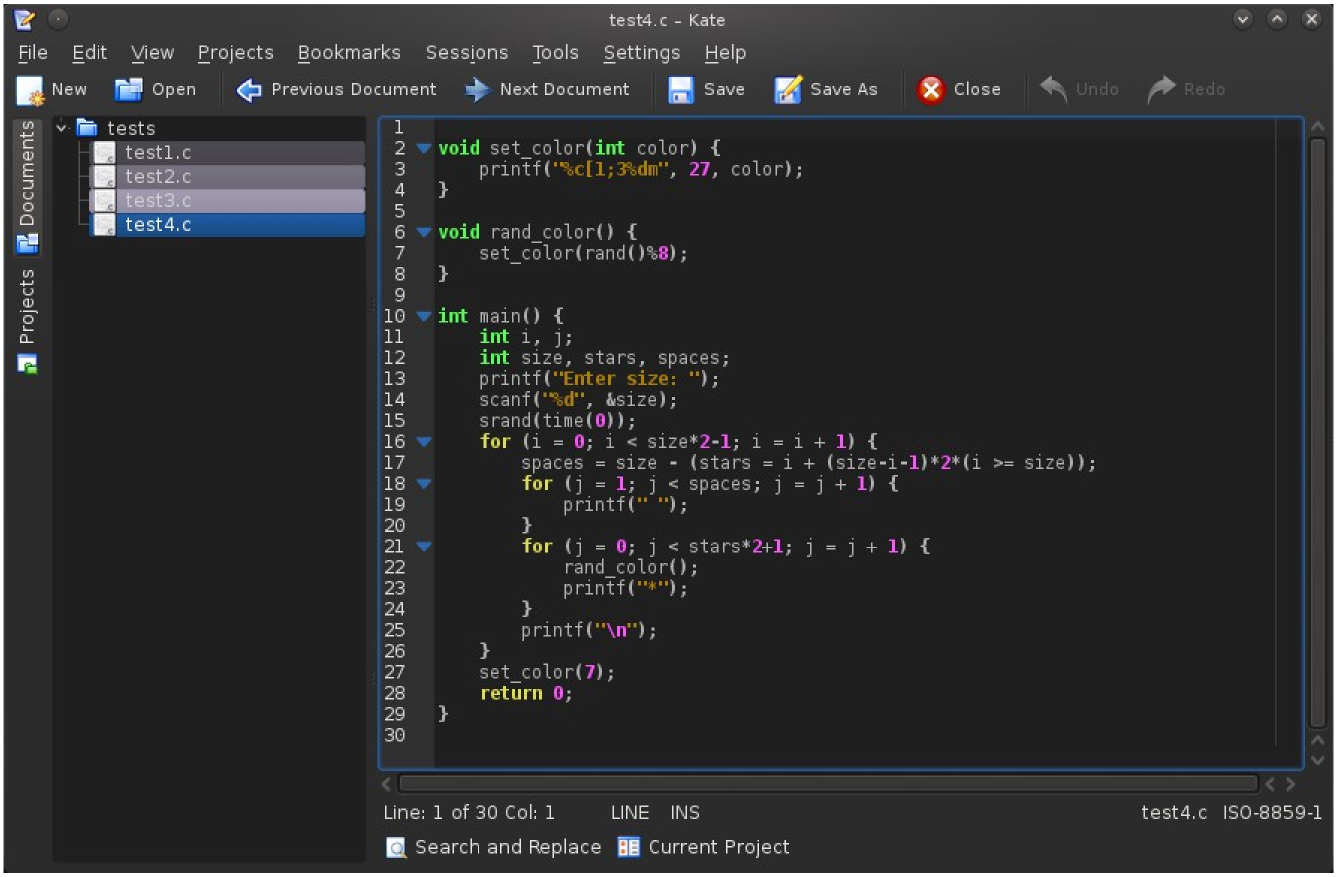

test3.c – Kate
File Edit View Projects Bookmarks Sessions Tools Settings Help
New Open Previous Document Next Document Save Save As Close Undo Redo
Documents
Projects
tests
test1.c
test2.c
test3.c
test4.c
Line: 9 of 9 Col: 1 LINE INS test3.c ISO-8859-1
Search and Replace Current Project

```

int main() {
    int y;
    printf("Enter integer: ");
    scanf("%d", &y);
    printf("Read value: %d\n", y);
    return 0;
}

```

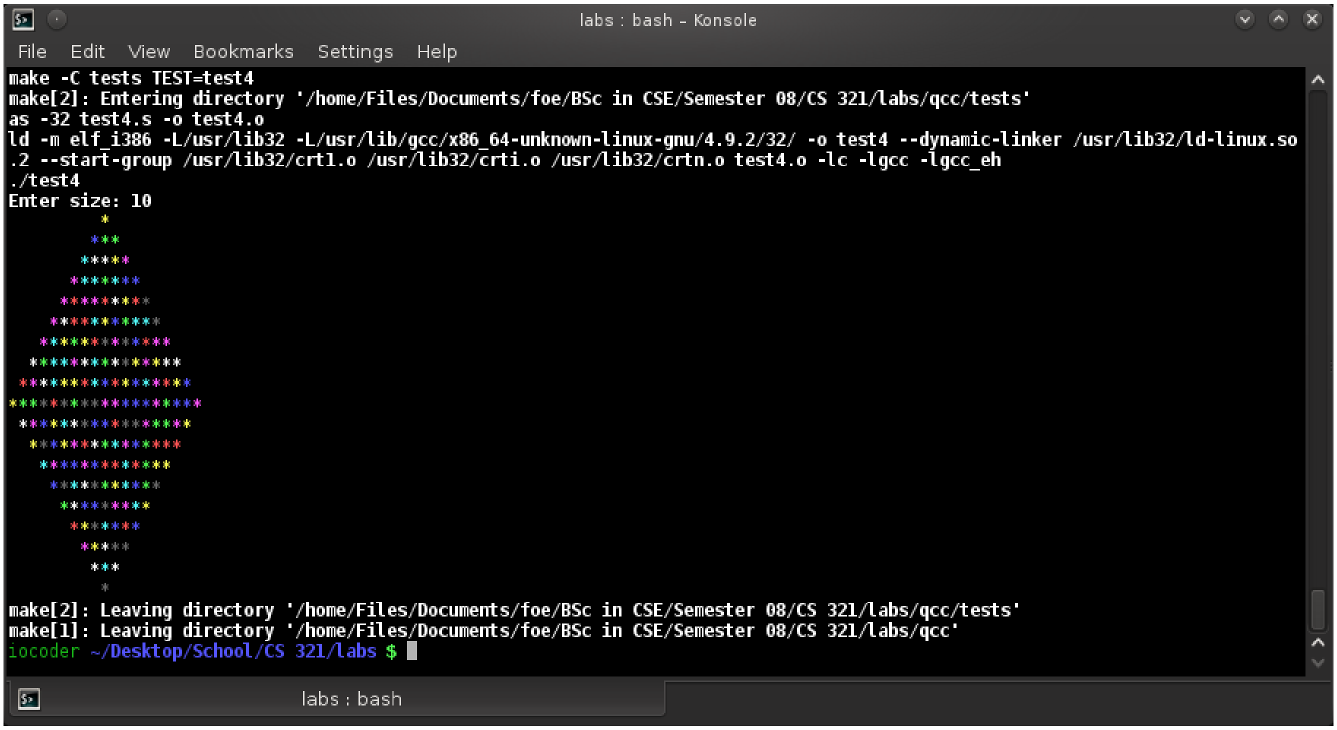


```
make[1]: Leaving directory '/home/Files/Documents/foe/BSc in CSE/Semester 08/CS 321/labs/leqs'
make -C qison
make[1]: Entering directory '/home/Files/Documents/foe/BSc in CSE/Semester 08/CS 321/labs/qison'
./bytearr.sh sim.c > sim_arr.c
cc -pedantic -Werror -include config.h -o qison ./product.c ./sim.c ./output.c ./ptable.c ./code.c ./token.c ./cfgfile.c ./bon
us.c ./main.c ./fillow.c ./sim_arr.c
make[1]: Leaving directory '/home/Files/Documents/foe/BSc in CSE/Semester 08/CS 321/labs/qison'
make -C qcc
make[1]: Entering directory '/home/Files/Documents/foe/BSc in CSE/Semester 08/CS 321/labs/qcc'
../leqs/leqs -n .
../qison/qison -n . yyout.c ./expr11.y ./expr14.y ./if.y ./main.y ./expr12.y ./symtab.y ./expr13.y ./expr04.y ./expr01.y ./exp
r10.y ./for.y ./expr02.y ./emit.y ./while.y ./expr06.y ./declr.y ./expr_stmt.y ./expr08.y ./expr15.y ./start.y ./expr00.y ./er
ror.y ./expr03.y ./expr05.y ./expr09.y ./expr07.y
cc -pedantic -Werror -std=c89 -Wno-variadic-macros -o qcc yyleqs.c yyout.c
make[1]: Leaving directory '/home/Files/Documents/foe/BSc in CSE/Semester 08/CS 321/labs/qcc'
make -C qcc run
make[1]: Entering directory '/home/Files/Documents/foe/BSc in CSE/Semester 08/CS 321/labs/qcc'
./qcc tests/test3.c tests/test3.s
make -C tests TEST=test3
make[2]: Entering directory '/home/Files/Documents/foe/BSc in CSE/Semester 08/CS 321/labs/qcc/tests'
as -32 test3.s -o test3.o
ld -m elf_i386 -L/usr/lib32 -L/usr/lib/gcc/x86_64-unknown-linux-gnu/4.9.2/32/ -o test3 --dynamic-linker /usr/lib32/ld-linux.so
.2 --start-group /usr/lib32/crt1.o /usr/lib32/crti.o /usr/lib32/crtn.o test3.o -lc -lgcc -lgcc_eh
./test3
Enter integer: -5
Read value: -5
make[2]: Leaving directory '/home/Files/Documents/foe/BSc in CSE/Semester 08/CS 321/labs/qcc/tests'
make[1]: Leaving directory '/home/Files/Documents/foe/BSc in CSE/Semester 08/CS 321/labs/qcc'
iocoder ~/Desktop/School/CS 321/labs $
```

Example 4:

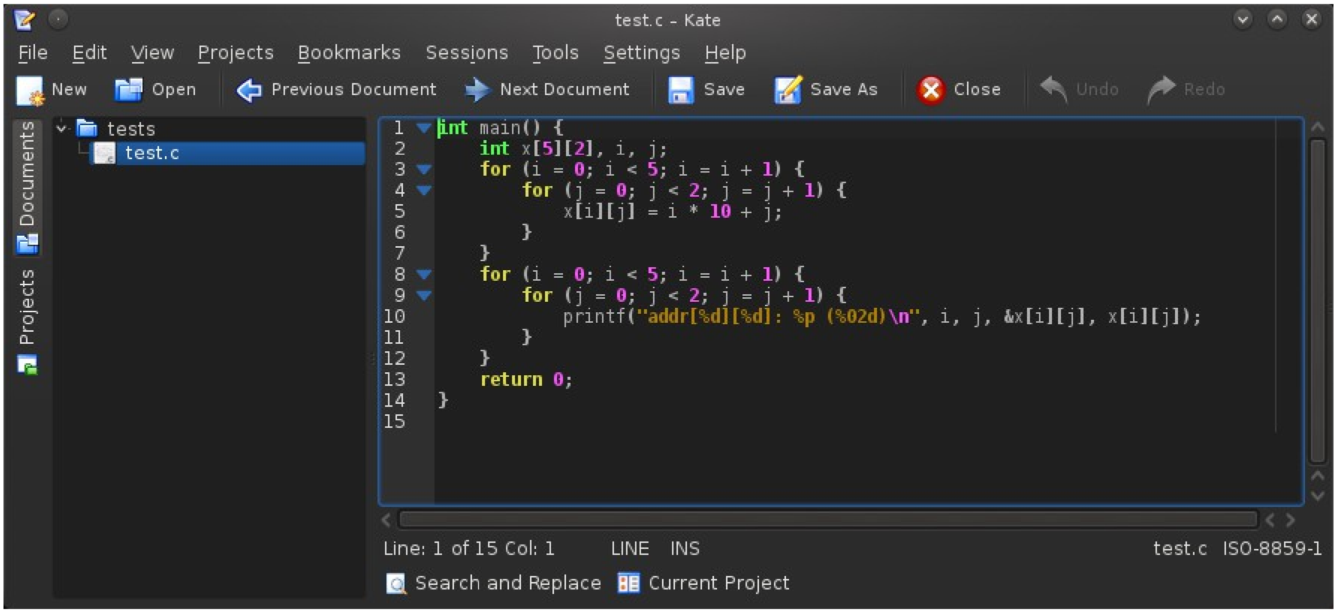


```c

void set_color(int color) {
    printf("%c[1;3%dm", 27, color);
}

void rand_color() {
    set_color(rand()%8);
}

int main() {
    int i, j;
    int size, stars, spaces;
    printf("Enter size: ");
    scanf("%d", &size);
    srand(time(0));
    for (i = 0; i < size*2-1; i = i + 1) {
        spaces = size - (stars = i + (size-i-1)*2*(i >= size));
        for (j = 1; j < spaces; j = j + 1) {
            printf(" ");
        }
        for (j = 0; j < stars*2+1; j = j + 1) {
            rand_color();
            printf("*");
        }
        printf("\n");
    }
    set_color(7);
    return 0;
}

```

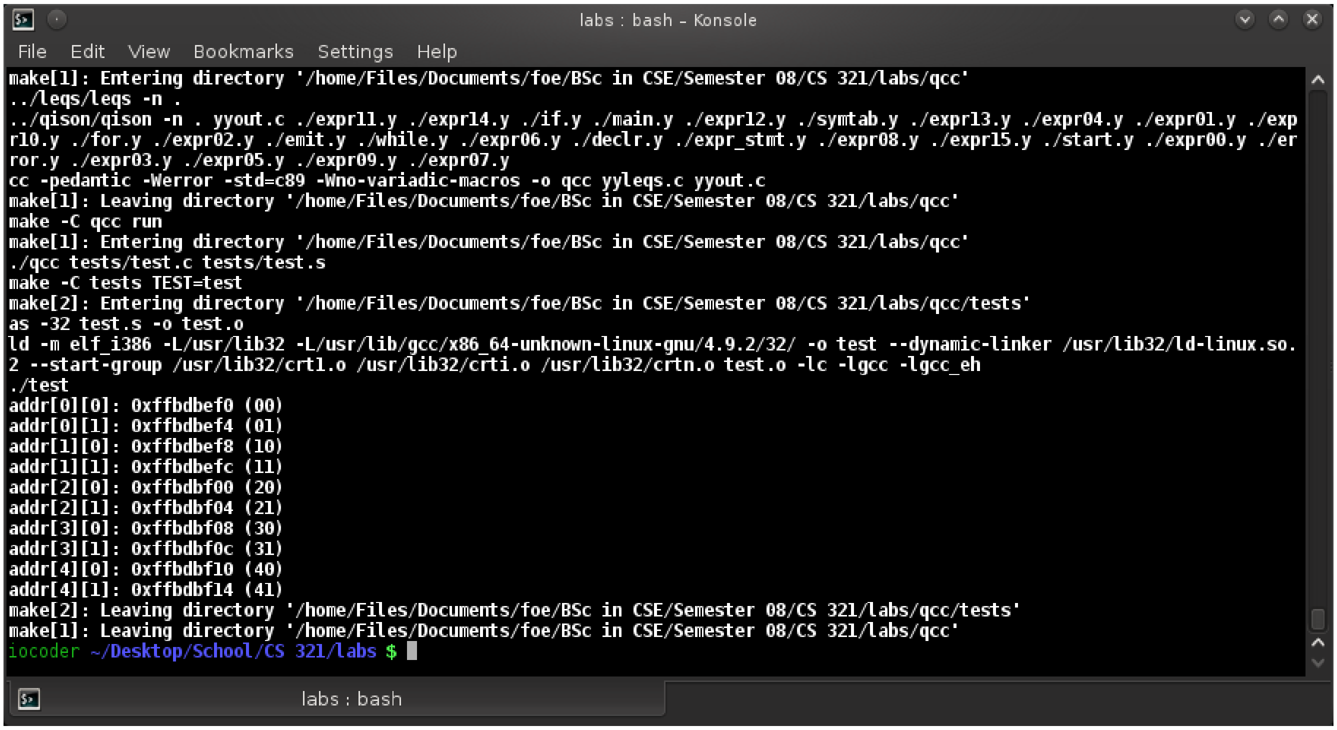


```
make -C tests TEST=test4
make[2]: Entering directory '/home/Files/Documents/foe/BSc in CSE/Semester 08/CS 321/labs/qcc/tests'
as -32 test4.s -o test4.o
ld -m elf_i386 -L/usr/lib32 -L/usr/lib/gcc/x86_64-unknown-linux-gnu/4.9.2/32/ -o test4 --dynamic-linker /usr/lib32/ld-linux.so
.2 --start-group /usr/lib32/crt1.o /usr/lib32/crti.o /usr/lib32/crtn.o test4.o -lc -lgcc -lgcc_eh
./test4
Enter size: 10
         *
        ***
       *****
      *******
     *********
    ***********
   *************
  ***************
 *****************
*******************
 *****************
  ***************
   *************
    ***********
     *********
      *******
       *****
        ***
         *
make[2]: Leaving directory '/home/Files/Documents/foe/BSc in CSE/Semester 08/CS 321/labs/qcc/tests'
make[1]: Leaving directory '/home/Files/Documents/foe/BSc in CSE/Semester 08/CS 321/labs/qcc'
iocoder ~/Desktop/School/CS 321/labs $
```

Example 5:


test.c – Kate
File Edit View Projects Bookmarks Sessions Tools Settings Help
New Open Previous Document Next Document Save Save As Close Undo Redo
Documents
Projects
tests
test.c
Line: 1 of 15 Col: 1 LINE INS test.c ISO-8859-1
Search and Replace Current Project

```c
int main() {
    int x[5][2], i, j;
    for (i = 0; i < 5; i = i + 1) {
        for (j = 0; j < 2; j = j + 1) {
            x[i][j] = i * 10 + j;
        }
    }
    for (i = 0; i < 5; i = i + 1) {
        for (j = 0; j < 2; j = j + 1) {
            printf("addr[%d][%d]: %p (%02d)\n", i, j, &x[i][j], x[i][j]);
        }
    }
    return 0;
}
```



```
make[1]: Entering directory '/home/Files/Documents/foe/BSc in CSE/Semester 08/CS 321/labs/qcc'
../leqs/leqs -n .
../qison/qison -n . yyout.c ./expr11.y ./expr14.y ./if.y ./main.y ./expr12.y ./symtab.y ./expr13.y ./expr04.y ./expr01.y ./exp
r10.y ./for.y ./expr02.y ./emit.y ./while.y ./expr06.y ./declr.y ./expr_stmt.y ./expr08.y ./expr15.y ./start.y ./expr00.y ./er
ror.y ./expr03.y ./expr05.y ./expr09.y ./expr07.y
cc -pedantic -Werror -std=c89 -Wno-variadic-macros -o qcc yyleqs.c yyout.c
make[1]: Leaving directory '/home/Files/Documents/foe/BSc in CSE/Semester 08/CS 321/labs/qcc'
make -C qcc run
make[1]: Entering directory '/home/Files/Documents/foe/BSc in CSE/Semester 08/CS 321/labs/qcc'
./qcc tests/test.c tests/test.s
make -C tests TEST=test
make[2]: Entering directory '/home/Files/Documents/foe/BSc in CSE/Semester 08/CS 321/labs/qcc/tests'
as -32 test.s -o test.o
ld -m elf_i386 -L/usr/lib32 -L/usr/lib/gcc/x86_64-unknown-linux-gnu/4.9.2/32/ -o test --dynamic-linker /usr/lib32/ld-linux.so.
2 --start-group /usr/lib32/crt1.o /usr/lib32/crti.o /usr/lib32/crtn.o test.o -lc -lgcc -lgcc_eh
./test
addr[0][0]: 0xffbdbef0 (00)
addr[0][1]: 0xffbdbef4 (01)
addr[1][0]: 0xffbdbef8 (10)
addr[1][1]: 0xffbdbefc (11)
addr[2][0]: 0xffbdbf00 (20)
addr[2][1]: 0xffbdbf04 (21)
addr[3][0]: 0xffbdbf08 (30)
addr[3][1]: 0xffbdbf0c (31)
addr[4][0]: 0xffbdbf10 (40)
addr[4][1]: 0xffbdbf14 (41)
make[2]: Leaving directory '/home/Files/Documents/foe/BSc in CSE/Semester 08/CS 321/labs/qcc/tests'
make[1]: Leaving directory '/home/Files/Documents/foe/BSc in CSE/Semester 08/CS 321/labs/qcc'
iocoder ~/Desktop/School/CS 321/labs $ 
```

THANK YOU