**Task 1: calc.c0**

/\*

\* Clac, stack-based calculator language

\*

\* 15-122 Principles of Imperative Computation

\*/

#use <string>

#use <parse>

#use <conio>

#use <args>

#use <util>

/\* check if stack is empty \*/

void check(stack S) {

if (stack\_empty(S)) {

/\* stack is empty \*/

error("Stack is too short to opearte");

}

}

/\* Return true: clac top-level interpreter will expect more input \*/

/\* Return false: clac top-level interpreter will exit \*/

bool eval(queue Q, stack S)

//@ensures \result == false || queue\_empty(Q);

{

while (!queue\_empty(Q)) {

string tok = deq(Q); /\* is this dequeue safe? \*/

// print("Read: "); print(tok); print("\n");

if (string\_equal(tok, "print")) {

/\* next line is unsafe and should be fixed \*/

check(S);

int x = pop(S);

printint(x); print("\n");

} else if (string\_equal(tok, "quit")) {

return false; /\* do not continue \*/

} else if (string\_equal(tok, "+")) {

check(S);

int y = pop(S);

check(S);

int x = pop(S);

push(S, x+y);

} else if (string\_equal(tok, "-")) {

check(S);

int y = pop(S);

check(S);

int x = pop(S);

push(S, x-y);

} else if (string\_equal(tok, "\*")) {

check(S);

int y = pop(S);

check(S);

int x = pop(S);

push(S, x\*y);

} else if (string\_equal(tok, "/")) {

check(S);

int y = pop(S);

check(S);

int x = pop(S);

if (y == 0) {

error("Division by zero");

}else if (x == int\_min() && y == -1) {

error("Overflow");

}else{

push(S, x/y);

}

} else if (string\_equal(tok, "%%")) {

check(S);

int y = pop(S);

check(S);

int x = pop(S);

if (y == 0) {

error("Division by zero");

}else if (x == int\_min() && y == -1) {

error("Overflow");

}else {

push(S, x%y);

}

} else if (string\_equal(tok, "<")) {

check(S);

int y = pop(S);

check(S);

int x = pop(S);

if (x < y) {

push(S, 1);

}else {

push(S, 0);

}

} else if (string\_equal(tok, "drop")) {

check(S);

pop(S);

} else if (string\_equal(tok, "dup")) {

check(S);

int x = pop(S);

push(S, x);

push(S, x);

} else if (string\_equal(tok, "swap")) {

check(S);

int y = pop(S);

check(S);

int x = pop(S);

push(S, y);

push(S, x);

} else if (string\_equal(tok, "rot")) {

check(S);

int z = pop(S);

check(S);

int y = pop(S);

check(S);

int x = pop(S);

push(S, z);

push(S, y);

push(S, x);

} else {

/\* not defined as an operation name, should be an int \*/

int\* p = parse\_int(tok, 10);

if (p == NULL) { /\* not an int \*/

/\* call error(msg) which prints msg and aborts \*/

error(string\_join("undefined token ", tok));

}

push(S, \*p);

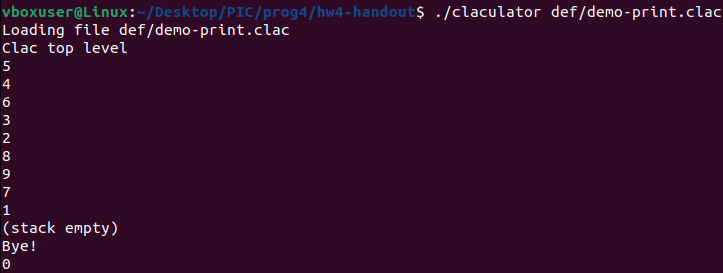
}

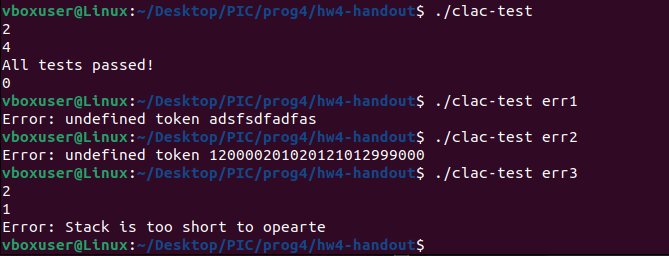
}

return true; /\* continue \*/

}

测试结果：





Task2: clac.c0

/\*

\* Clac, stack-based calculator language

\*

\* 15-122 Principles of Imperative Computation

\*/

#use <string>

#use <parse>

#use <conio>

#use <args>

#use <util>

/\* check if stack is empty \*/

void check(stack S) {

if (stack\_empty(S)) {

/\* stack is empty \*/

error("Stack is too short to opearte");

}

}

/\* Return true: clac top-level interpreter will expect more input \*/

/\* Return false: clac top-level interpreter will exit \*/

bool eval(queue Q, stack S)

//@ensures \result == false || queue\_empty(Q);

{

int skip = 0;

while (!queue\_empty(Q)) {

string tok = deq(Q); /\* is this dequeue safe? \*/

// print("Read: "); print(tok); print("\n");

if (skip > 0) {

/\* skip n times \*/

skip--;

}else {

if (string\_equal(tok, "print")) {

/\* next line is unsafe and should be fixed \*/

check(S);

int x = pop(S);

printint(x); print("\n");

} else if (string\_equal(tok, "quit")) {

return false; /\* do not continue \*/

} else if (string\_equal(tok, "+")) {

check(S);

int y = pop(S);

check(S);

int x = pop(S);

push(S, x+y);

} else if (string\_equal(tok, "-")) {

check(S);

int y = pop(S);

check(S);

int x = pop(S);

push(S, x-y);

} else if (string\_equal(tok, "\*")) {

check(S);

int y = pop(S);

check(S);

int x = pop(S);

push(S, x\*y);

} else if (string\_equal(tok, "/")) {

check(S);

int y = pop(S);

check(S);

int x = pop(S);

if (y == 0) {

error("Division by zero");

}else if (x == int\_min() && y == -1) {

error("Overflow");

}else{

push(S, x/y);

}

} else if (string\_equal(tok, "%%")) {

check(S);

int y = pop(S);

check(S);

int x = pop(S);

if (y == 0) {

error("Division by zero");

}else if (x == int\_min() && y == -1) {

error("Overflow");

}else {

push(S, x%y);

}

} else if (string\_equal(tok, "<")) {

check(S);

int y = pop(S);

check(S);

int x = pop(S);

if (x < y) {

push(S, 1);

}else {

push(S, 0);

}

} else if (string\_equal(tok, "drop")) {

check(S);

pop(S);

} else if (string\_equal(tok, "dup")) {

check(S);

int x = pop(S);

push(S, x);

push(S, x);

} else if (string\_equal(tok, "swap")) {

check(S);

int y = pop(S);

check(S);

int x = pop(S);

push(S, y);

push(S, x);

} else if (string\_equal(tok, "rot")) {

check(S);

int z = pop(S);

check(S);

int y = pop(S);

check(S);

int x = pop(S);

push(S, z);

push(S, y);

push(S, x);

} else if (string\_equal(tok, "skip")) {

check(S);

int x = pop(S);

if (x < 0) {

error("n must be not negative");

}

skip = x;

} else if (string\_equal(tok, "if")) {

check(S);

int x = pop(S);

if (x == 0) {

skip = 2;

}

} else if (string\_equal(tok, "else")) {

skip = 1;

} else if (string\_equal(tok, "pick")) {

check(S);

int x = pop(S);

if (x <= 0) {

error("n must be positive");

}

stack new = stack\_new();

for (int i = 0; i < x - 1; i++) {

check(S);

int j = pop(S);

push(new, j);

}

check(S);

int y = pop(S);

push(new, y);

for (int i = 0;i < x - 1; i++) {

int j = pop(new);

push(S, j);

}

push(S ,y);

}

else {

/\* not defined as an operation name, should be an int \*/

int\* p = parse\_int(tok, 10);

if (p == NULL) { /\* not an int \*/

/\* call error(msg) which prints msg and aborts \*/

error(string\_join("undefined token ", tok));

}

push(S, \*p);

}

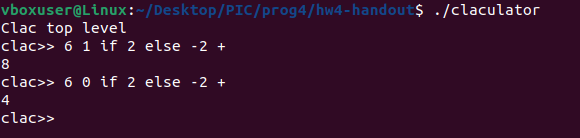
}

}

return true; /\* continue \*/

}

测试结果：



Task 3:

指令：copy

作用：将栈内的数字均复制一个到相邻位置上

举例：S，1,2,3 || copy，Q 🡪 S，1,1,2,2,3,3 || Q

新增代码：

…

else if (string\_equal(tok, "copy")) {

stack new = stack\_new();

while (!stack\_empty(S)) {

int x = pop(S);

push(new, x);

push(new, x);

}

while (!stack\_empty(new)) {

int x = pop(new);

push(S, x);

}

}

…

测试结果：

