帮助 清除

10*5 = 50, 0x032, 110010

计算

你能用它们算出24吗?

5 2 3 7

看答案 下一个

5 - 2 + 3 * 7

随笔分类

C/C++(42)
Great CPUs(13)
HTML/JS(18)
misc(318)
Python(40)
sqlite(9)
suanpan(61)
唉(30)
狄更斯的英国史(238)
六级/考研词汇(666)
四级词汇(621)
象棋(26)

基本能看懂的C编译器，只有361行!

Fabrice Bellard is a French computer programmer known for writing FFmpeg, QEMU, and the Tiny C Compiler 和许许多多别的。

我修改了下他写的otccelfn.c，增加/删除了一点功能，提高了可读性。它能编译像下面这样的C程序：

```
gcd(long a, long b) {
  long i;
  i = a - b;
  if (i == 0) return a;
  else if (i > 0) return gcd(i, b);
  else return gcd(a, -i);
}
tonum(long s) {
  long n, i;
  for (n = 0; ;s++) {
    if ((i = *(char*)s) == 0) break;
    n = n * 10 + (i - '0');
  }
  return n;
}
main() {
  long i, j;
  i = tonum("20"); j = tonum("24");
  printf("%ld\n", gcd(i, j));
  return 0;
}
```

说明:

- 不支持注释和宏；忽略#include【删除功能以使程序变短】
- 【增加】函数参数类型为long. x64下int是32位，long和void*都是64位，x86下它们都是32位，因此解除了只能-m32的限制。
  - 原版：没有类型就是int，64位下char*当int传就错了。
- 数据类型只能用long，但编译器里还叫T_INT.
- 原版和修改版都不支持 op= 如 i += 3; 而且都不支持数组，do和switch等等——太多了列不完。
- Tiny C Compiler功能很完备。wc -l *.c *h 43555行。如果把arm-gen, arm64-asm, i386*, x86_64*等都去掉，改成bytecode和虚拟机，估计3万多行。configure贼快，config.h只有5行，就定义了DIR和VERSION.
- gcd是Greatest Common Divisor (最大公约数)的缩写。

编译器的全部源码：

```
// Adapted from otccelfn.c (written by Fabrice Bellard)
#include <assert.h>
#include <stdio.h>
#include <ctype.h>
#pragma warning(disable:4786)
#include <vector>
#include <string>
#include <map>
using namespace std;
typedef unsigned char byte;
```

```
typedef unsigned      uint;

////////////////////// lexer //////////////////////////
FILE* srcfp;
enum { T_NUM=1, T_DOUBLE, T_INT='i', T_IF, T_ELSE, T_WHILE, T_FOR, T_BREAK, T_RETURN
struct opr {
  uint code;
  int ws; // weird stuff
  char ch, ch2;
} oprs[] = { // https://en.cppreference.com/w/c/language/operator_precedence
  { 0x1, 11, '+', '+' },
  { 0xff, 11, '-', '-' },
  { 0x1, 10, '|', '|' },
  { 0x0, 9, '&', '&' },
  { 0xc809, 8, '|' },
  { 0xc831, 7, '^' },
  { 0xc821, 6, '&' },
  { 0x4, 5, '=', '=' },
  { 0x5, 5, '!', '=' },
  { 0xe, 4, '<', '=' },
  { 0xd, 4, '>', '=' },
  { 0xc, 4, '<' },
  { 0xf, 4, '>' },
  { 0xe0d391, 3, '<', '<' },
  { 0xd391, 3, '>', '>' },
  { 0xc801, 2, '+' },
  { 0xd8f7c829, 2, '-' }, // f7 d8 is neg %eax; 29 c8 is sub %ecx,%eax
  { 0xd0f7, 2, '~' },
  { 0x4, 2, '!' },
  { 0xc1af0f, 1, '*' }, // 0f af c1 imul %ecx, %eax
  { 0xf9f79991, 1, '/' },
  { 0xf9f79991, 1, '%' },
  { 0x0, 0, '*' },  // typecast
};
typedef map<string, int> Str2N;
Str2N keywords;
struct Symbols {
  vector<int> addr; // address = [id]
  Str2N str2n;  // id = [name]
  int add(const string& name) {
    int id;
    Str2N::const_iterator p = str2n.find(name);
    if (p == str2n.end()) { str2n[name] = id = addr.size(); addr.push_back(0); }
    else id = p->second;
    return id;
  }
  int& operator[](int id) { return addr[id]; }
  int& operator[](const string& name) { return addr[str2n[name]]; }
} symbols;
char _ch;
int tk, tkval, tkws; string tkstr;

inline bool is_alnum(int c) { return c == '_' || isalnum(c); }

inline int read_char() {
  if ((_ch = getc(srcfp)) == '#') do _ch = getc(srcfp); while (_ch != '\n');
  return _ch;
}

int read_char_unesc() {
  if (_ch != '\\') return _ch;
  if (read_char() == 'n') return _ch = '\n';
  else assert(!"correct escape sequence");
  return 0;
}

void read_token() { // _ch should already be read
  tk = tkval = tkws = 0; tkstr = ""; // clear token
  while (isspace(_ch)) read_char(); // skip white spaces
  if (is_alnum(_ch)) {
    do tkstr += _ch; while (is_alnum(read_char()));
    if (isdigit(tkstr[0])) { tk = T_NUM; tkval = atoi(tkstr.c_str()); return; } // N
```

```
    Str2N::iterator p = keywords.find(tkstr);
    if (p != keywords.end()) { tk = p->second; return; } // KEYWORD
    tk = T_SYMBOL; tkval = symbols.add(tkstr); return; // SYMBOL
  }
  char c = _ch; read_char(); // check two chars: c, _ch
  if (c == '\'') { // '0' ' ' etc
    tk = T_NUM; // they are also numbers, e.g ' ' is 32
    tkstr = tkval = read_char_unesc();
    if (read_char() != '\'') assert(!" is '\''");
    read_char(); return; // NUMBER
  }
  tkstr = tk = c;
  for (const opr* p = oprs; p < &oprs[sizeof(oprs) / sizeof(oprs[0])]; ++p) {
    if (p->ch == c && (p->ch2 == _ch || !p->ch2)) {
      tkval = p->code; tkws = p->ws;
      if (p->ch2 == _ch) { tk = T_DOUBLE; tkstr += _ch; read_char(); return; } // ++
      break;
    }
  }
  // ( ) + * and so on. parser handles strings
}

int next_tk() {
  read_token();
  return tk;
  if (tk == EOF) printf("EOF\n");
  else if (tk == T_NUM) printf("%d\n", tkval);
  else if (tk == T_DOUBLE) printf("%s\n", tkstr.c_str());
  else if (tk >= T_SYMBOL) printf(".%d `%s'\n", tkval, tkstr.c_str());
  else if (tk >= T_INT) printf("%s\n", tkstr.c_str());
  else printf("%c\n", tk);
  return tk;
}

#define curr_must_be(c) (assert(tk == c), next_tk())
#define next_must_be(c) (next_tk(), assert(tk == c), next_tk())

////////////////////////// yacc/yuck? :-) ////////////////////////////
#define STARTUP_CODE_SIZE 7 // startup code calls main and then exit
#define FIRST_GLOBAL_VAR_ADDR 4 // don't use 0 -- what about NULL?
#define DATA_SECTION_START 0x6800 // loader puts the data section here
byte code[4096], data[4096];
uint code_end = STARTUP_CODE_SIZE, data_end = FIRST_GLOBAL_VAR_ADDR;

void modify32(uint i, int n) { // code[i] = n; little endian
  byte* p = code + i; *p = n; p[1] = n >> 8; p[2] = n >> 16; p[3] = n >> 24;
}

void out_4(uint n) {
  byte* p = code + code_end;
  *p = n; p[1] = n >> 8; p[2] = n >> 16; p[3] = n >> 24;
  code_end += 4;
}

void out_vl(uint n) { for (;n;n >>= 8) code[code_end++] = n; } // variable length

uint gen(uint a, uint n) { out_vl(a); uint r = code_end; out_4(n); return r; }

void gen_load_imm(uint x) { gen(0xb8, x); } // b8 ?? ?? ?? ?? mov $0x0,%eax

inline uint gen_jmp(uint to) { return gen(0xe9, to); }

void patch_jmp(uint i) {
  while (i) { // it's a chain!
    const byte* p = code + i;
    uint next = *p | (p[1] << 8) | (p[2] << 16) | (p[3] << 24);
    modify32(i, code_end - i - 4);
    i = next;
  }
}

uint gen_je_or_jne(int i, int to) { // i: 0=je, 1=jne
```

```
  out_vl(0x0fc085); // 85 c0 test %eax, %eax; 0f 84 ?? ?? ?? ?? je
  return gen(0x84 + i, to);
}

void gen_cmp(int i) {
  out_vl(0xc139);   // 39 c1    cmp %eax,%ecx
  out_vl(0xc031);   // 31 c0    xor %eax, %eax
  out_vl(0x0f);     // 0f 94 c0 sete %al ; set_if_equal
  out_vl(0x90 + i); // 0f 9f c0 setg %al ; set_if_greater
  out_vl(0xc0);     // 85 c0    test   %eax,%eax
}

// 83 85 08 00 00 00 01 addl $0x1,0x8(%ebp)
// 89 85 fc ff ff ff    mov  %eax,-0x4(%ebp
// 8b 85 08 00 00 00    mov  0x8(%ebp),%eax
// 8d 85 ?? ?? ?? ?? https://pdos.csail.mit.edu/6.828/2006/readings/i386/LEA.htm
void gen_add_mov_lea(int i, int addr) { out_vl(i + 0x83); gen(0x85, addr); }

int ret_point, local;
void _expr(int);
#define do_expr() _expr(11) // 11 is ++ and --

// deal with i=  a-  b;  i==  0)  i>  gcd(  i,  b)  a,  -i and so on
void do_2_tokens(bool do_assign) {
  if (tk == '\"') { // string
    gen_load_imm(DATA_SECTION_START + data_end);
    while (_ch != '\"') { read_char_unesc(); data[data_end++] = _ch; read_char(); }
    data[data_end] = 0;
    data_end = (data_end + 4) & ~3; // align heap -4=~3
    read_char(), next_tk(); return;
  }

  // save useful information of current token before calling next_tk
  int saddr = (tk == T_SYMBOL) ? symbols[tkval] : 0;
  int tt = tk;  // token ype
  int tkv = tkval;  // token value
  int ws = tkws;
  string  tks = tkstr;
  next_tk();
  //printf("2 tokens: %s%s\n", tks.c_str(), tkstr.c_str());

  if (tt == T_NUM) gen_load_imm(tkv);
  else if (ws == 2) { // + - ! ~
    do_2_tokens(false);
    gen(0xb9, 0); // movl $0, %ecx
    if (tt == '!')  // e.g. !i, to know !i is to know whether i equals 0
      gen_cmp(tkv); // that is to compare i and 0, and that's substraction
    else out_vl(tkv);
  } else if (tt == '(') { do_expr(); curr_must_be(')'); }
  else if (tt == '*') { // typecast e.g. *(char*)
    curr_must_be('('), next_tk(), next_tk(); curr_must_be(')');
    do_2_tokens(false);
    if (tk == '=') {
      next_tk();
      out_vl(0x50); // push %eax
      do_expr();
      out_vl(0x59); // pop %ecx
      out_vl(0x0188 + (tt == T_SYMBOL)); // movl %eax/%al, (%ecx)
    } else if (tt) {
      if (tt == T_INT) out_vl(0x8b); // mov (%eax), %eax
      else { out_vl(0xbe0f); code[code_end++] = 0; } // movsbl (%eax),%eax
    }
  } else if (tt == '&') { // lea imm(%ebp) %eax
    gen_add_mov_lea(10, saddr); next_tk();
  }
  else if (tk == '(') { // function call
    int subesp = gen(0xec81, 0); // 81 ec 00 00 00 00 sub $0, %esp (to be modified)
    next_tk();
    int i = 0; for (;tk != ')'; i += 4) {
      do_expr(); gen(0x248489, i); // 89 84 24 ?? ?? ?? ?? movl %eax, xxx(%esp)
      if (tk == ',') next_tk();
    }
```

```
    modify32(subesp, i); // modify jmp sub $xxx, %esp
    curr_must_be(')');
    gen(0xe8, saddr - code_end - 5); // call
    if (i) gen(0xc481, i); // add $xxx, %esp
  }
  else if (tk == '=' && do_assign) {
    next_tk(); do_expr(); gen_add_mov_lea(6, saddr); // mov %eax, effective_address
  } else if (tk != '(') { // variable
    gen_add_mov_lea(8, saddr); /* mov EA, %eax */
    if (tkws == 11) { // ++ -- tkval is instruction
      gen_add_mov_lea(0, saddr); out_vl(tkval); next_tk();
    }
  }
}

void _expr(int what) {
  if (what-- == 1) do_2_tokens(true); // * / %
  else {
    _expr(what);
    uint pos = 0;
    int  tkv = tkval;
    while (what == tkws) {
      int tt = tk; tkv = tkval; next_tk();
      if (what > 8) { // && || -- ++
        pos = gen_je_or_jne(tkv, pos); _expr(what);
      } else {
        out_vl(0x50); _expr(what); out_vl(0x59); // push %eax; eval; pop %ecx
        if (what == 4 || what == 5) gen_cmp(tkv); // > < >= <= != ==
        else { // << >> + - ~
          out_vl(tkv);
          // _expr(2) -  _expr(1) - do_2_tokens(true), then here
          if (tt == '%') out_vl(0x92); // xchg %edx, %eax
        }
      }
    }
    if (pos && what > 8) { // && ||
      pos = gen_je_or_jne(tkv, pos);
      gen_load_imm(tkv ^ 1);
      gen_jmp(5); // jmp $ + 5
      patch_jmp(pos);
      gen_load_imm(tkv);
    }
  }
}

uint do_test_expr() { do_expr(); return gen_je_or_jne(0, 0); } // je 00 00 00 00

void do_block(uint* plevel) {
  uint  pos, p2, p3, tt;
  if (tk == T_IF) {
    next_must_be('('); pos = do_test_expr(); curr_must_be(')');
    do_block(plevel);
    if (tk == T_ELSE) {
      next_tk(); p2 = gen_jmp(0); patch_jmp(pos);
      do_block(plevel); patch_jmp(p2);
    }
    else patch_jmp(pos);
  }
  else if (tk == T_WHILE || tk == T_FOR) {
    tt = tk; next_must_be('(');
    if (tt == T_WHILE) { p2 = code_end; pos = do_test_expr(); }
    else { // for (i=0; i<9; i++) {}
      if (tk != ';') do_expr(); // for (;i<9 ...
      curr_must_be(';');
      p2 = code_end; pos = 0;
      if (tk != ';') pos = do_test_expr(); // for(;;i++ ...
      curr_must_be(';');
      if (tk != ')') { // for(;;i++)
        p3 = gen_jmp(0);
        do_expr();
        gen_jmp(p2 - code_end - 5); patch_jmp(p3);
        p2 = p3 + 4;
```

```
      }
    }
    curr_must_be(')');
    do_block(&pos);
    gen_jmp(p2 - code_end - 5); patch_jmp(pos);
  }
  else if (tk == '{') { // declaration
    for (next_tk(); tk == T_INT;) { // int i; int j, k;
      for (next_tk(); tk == T_SYMBOL;) {
        symbols[tkval] = -(local += 4);
        if (next_tk() == ',') next_tk();
      }
      curr_must_be(';'); // TODO: `int ;'
    }
    while(tk != '}') do_block(plevel);
    next_tk();
  }
  else {
    if (tk == T_RETURN) {
      if (next_tk() != ';') do_expr();
      ret_point = gen_jmp(ret_point);
    }
    else if (tk == T_BREAK) { next_tk(); *plevel = gen_jmp(*plevel); }
    else if (tk != ';') do_expr();
    curr_must_be(';');
  }
}

void do_func() {
  symbols[tkval] = code_end; // put function address
  int n = 8; // stack: LOW ... ebp ret_addr params HIGH
  next_must_be('(');
  while (tk != ')') { // read param name and compute offset
    curr_must_be(T_INT);
    assert(tk == T_SYMBOL); symbols[tkval] = n; n += 4;
    if (next_tk() == ',') next_tk();
  } next_tk();
  out_vl(0xe58955); // 55 push %ebp; 89 e5 mov %esp, %ebp
  uint pos = gen(0xec81, 0); // sub imm, %esp
  ret_point = local = 0; do_block(0);
  patch_jmp(ret_point);
  out_vl(0xc3c9); // c9 leave; c3 ret
  modify32(pos, local); // save local variables
}

void save_bin_files() {
  FILE* fp;
  fp = fopen("data.bin", "wb"); fwrite((void*)data, 1, data_end, fp); fclose(fp);
  int csize = code_end; code_end = 0; // add the startup code
  out_vl(0xe8); out_4(symbols["main"] - 5); // e8 ?? ?? ?? ?? call
  out_vl(0x80cd); // int $0x80
  fp = fopen("code.bin", "wb"); fwrite((void*)code, 1, csize, fp); fclose(fp);
}

int main() {
  srcfp = fopen("gcd.c", "rt"); assert(srcfp);
  Str2N& kw = keywords; kw["long"]=T_INT; kw["if"]=T_IF; kw["else"]=T_ELSE;
  kw["while"]=T_WHILE; kw["for"]=T_FOR; kw["break"]=T_BREAK; kw["return"]=T_RETURN;
  read_char(), next_tk(); // parser wants a token; lexer wants a char
  while (tk != EOF) do_func();
  fclose(srcfp), save_bin_files();
  return 0;
}
```

编译器里：srcfp = fopen("gcd.c", "rt"); assert(srcfp); 这当然很好改。

编译器的输出是code.bin和data.bin，也写死了，也好改。code.bin里是x86 32位机器码，在Linux下，可用objdump反汇编查看：

```
objdump -b binary -m i386 -D code.bin
```

如：

```
   0: e8 18 00 00 00        call   0x1d
   5: cd 80                 int    $0x80
   7: 55                    push   %ebp
   8: 89 e5                 mov    %esp,%ebp
   a: 81 ec 00 00 00 00     sub    $0x0,%esp
  10: b8 03 00 00 00        mov    $0x3,%eax
  15: 89 85 08 00 00 00     mov    %eax,0x8(%ebp)
  1b: c9                    leave
  1c: c3                    ret
  1d: 55                    push   %ebp
  1e: 89 e5                 mov    %esp,%ebp
  20: 81 ec 04 00 00 00     sub    $0x4,%esp
  26: b8 04 00 00 00        mov    $0x4,%eax
  2b: 89 85 fc ff ff ff     mov    %eax,-0x4(%ebp)
  31: 81 ec 04 00 00 00     sub    $0x4,%esp
  37: 8b 85 fc ff ff ff     mov    -0x4(%ebp),%eax
  3d: 89 84 24 00 00 00 00  mov    %eax,0x0(%esp)
  44: e8 be ff ff ff        call   0x7
  49: 81 c4 04 00 00 00     add    $0x4,%esp
  4f: c9                    leave
  50: c3                    ret
```

下面是个玩具虚拟机的全部代码(97行)：

```
#include <stdio.h>
#include <stdlib.h>
typedef unsigned char byte;

byte  mem[65536];
void load(int i) {
  FILE* fp = fopen(i ? "data.bin" : "code.bin", "rb");
  fseek(fp, 0, SEEK_END); int n = ftell(fp); fseek(fp, 0, SEEK_SET);
  fread(mem + i, 1, n, fp); fclose(fp);
}
int r32(byte* p) { return *p | (p[1] << 8) | (p[2] << 16) | (p[3] << 24); } // littl
int r32(int a) { return r32(mem + a); }
void w32(int a, int v) { *((int*)&mem[a]) = v; }

int reg[16];
#define eax reg[0]
#define ecx reg[2]
#define edx reg[3]
#define res reg[10]
#define eip reg[11]
#define ebp reg[14]
#define esp reg[15]
void set_al(int c) { eax = (eax & 0xffffff00) | (c ? 1 : 0); }

void print() {
  printf("%3x: %2x | ax %8x cx %8x dx %4x bp %4x sp %4x | -4: ", eip, mem[eip], eax,
  for (int j = esp - 4; j < esp + 12; j++) printf(mem[j] ? "%02x " : "   ", mem[j]);
  puts("+c");
  //getchar();
}
void quit() { printf("Press Enter to quit."); getchar(); exit(0); }
#define _(i) break;

int main() {
  load(0), load(0x6800);
  for (esp = 65532;;) {
  int oip = eip, i, j;
  byte op = mem[eip], *p = &mem[eip];
  print();
```

```
  switch (op) {
  case 0x01: eax += ecx; eip += 2; _(add %ecx %eax)
  case 0x0f:
    if (p[1] == 0x84) { eip += 6; if (res == 0) eip += r32(p+2); }
    else if (p[1] == 0x9f) { set_al(res > 0); eip += 3; }
    else if (p[1] == 0x94) { set_al(res == 0); eip += 3; }
    else if (p[1] == 0xaf) { eax *= ecx; eip += 3; }
    else if (p[1] == 0xbe) { eax = char(mem[eax]); eip += 3; }
    break;
  case 0x29: eax -= ecx; eip += 2; _(sub %ecx %eax)
  case 0x31: eax ^= eax; eip += 2; _(xor %eax %eax)
  case 0x39: res = ecx - eax; eip += 2; _(cmp %eax %ecx)
  case 0x50: w32(esp -= 4, eax); ++eip; _(push %eax)
  case 0x55: w32(esp -= 4, ebp); ++eip; _(push %ebp)
  case 0x59: ecx = r32(esp); esp += 4; ++eip; _(pop %ecx)
  case 0x81:
    if (p[1] == 0xec) { esp -= r32(p+2); eip += 6; }
    else if (p[1] == 0xc4) { esp += r32(p+2); eip += 6; }
    break;
  case 0x83: i = r32(p+2)+ebp; w32(i, r32(i)+p[6]); eip += 7; _(addl imm imm(%ebp))
  case 0x85: res = eax & eax; eip += 2; _(test %eax %eax)
  case 0x89:
    if (p[1] == 0xe5) { ebp = esp; eip += 2; } // mov %esp,%ebp
    else if (p[1] == 0x84) { w32(esp + r32(p+3), eax); eip += 7; } // mov %eax,imm(%
    else if (p[1] == 0x85) { w32(ebp + r32(p+2), eax); eip += 6; } // mov %eax,imm(%
    break;
  case 0x8b:
    if (p[1] == 0x85) { eax = r32(ebp + r32(p+2)); eip += 6; } // mov imm(%ebp),%eax
    break;
  case 0x8d: eax = r32(p+2) + ebp; eip += 6; _(lea imm(%ebp) %eax)
  case 0x91: i = eax; eax = ecx; ecx = i; ++eip; _(xchg %eax %ecx)
  case 0x92: i = eax; eax = edx; edx = i; ++eip; _(xchg %eax %edx)
  // cltd converts the signed long in EAX to a signed double long in EDX:EAX
  case 0x99: edx = (eax < 0) ? -1 : 0; ++eip; _(cltd)
  case 0xb8: eax = r32(p+1); eip += 5; _(mov imm %eax)
  case 0xb9: ecx = r32(p+1); eip += 5; _(mov imm %ecx)
  case 0xc3: { eip = r32(esp); esp += 4; } _(ret)
  case 0xc9: { ebp = r32(esp = ebp); esp += 4; ++eip;} _(leave)
  case 0xcd: quit();
  case 0xe8:
    eip += 5; // return to next instruction
    j = eip + (i = r32(p + 1));
    // callee: stack: LOW local_vars ebp ret_addr param1 param2 HIGH
    // caller: sub $8, %esp; movl %eax, 0(%esp); movl %eax, 4(%esp); add $8, %esp
    // cdecl: caller cleans stack; fastcall, pascal, WINAPI
    if (!j) puts(""), printf((char*)&mem[r32(esp)], r32(esp + 4)), puts("");
    else { w32(esp -= 4, eip); eip += i; }
    break;
  case 0xe9: eip += 5 + r32(p + 1); _(jmp)
  case 0xf7:
    if (p[1] == 0xd8) eax = -eax; // neg %eax
    else if (p[1] == 0xf9 && ecx) { edx = eax % ecx; eax /= ecx; } // idiv %ecx
    eip += 2; break;
  }
  if (oip == eip) quit();
  }
  return 0;
}
```

它读入code.bin和data.bin并解释执行，输出像这样：

```
205: e8 | ax        4 cx        0 dx    0 bp fff4 sp ffe4 | -4: 18          0c 68

4

20a: 81 | ax        4 cx        0 dx    0 bp fff4 sp ffe4 | -4: 18          0c 68
210: b8 | ax        4 cx        0 dx    0 bp fff4 sp ffec | -4: 04          18
215: e9 | ax        0 cx        0 dx    0 bp fff4 sp ffec | -4: 04          18
```

```
21a: c9 | ax        0 cx        0 dx    0 bp fff4 sp ffec | -4: 04          18
21b: c3 | ax        0 cx        0 dx    0 bp    0 sp fff8 | -4:             05
  5: cd | ax        0 cx        0 dx    0 bp    0 sp fffc | -4: 05
```

- 以上代码都既可以用Visual C++ 6，又可以用相当新的gcc/g++ (如version 10.2.1 20210110)编译。
- 有个bug: printf("%ld\n", 2+3-9); 输出结果不对，正在排查。
  - (2+3)-9是对的。如果我是老师就好了，可以当作业布置给学生：你说你看懂了？fix the bug to prove it. :-)
  - 2+3*4, 2*3+4, 2*(3+4)都是对的。原版是如何处理运算符优先级的，我没看明白。大胆地怀疑下原版就有处理同优先级运算符不对这个bug.
    - 实验表明原版没错，是我改坏了，已改正。
  - _expr里while (what == tkws)改成if (what == tkws)是我改坏的，结果好像是数不对局部变量的个数，已改正。
- 如何改成用bison？生成语法树先？总不能YYSTYPE里有个vector放已生成的代码吧？gawk用的是链表。C++里vector赋值，可以swap而不是复制一遍。还可以用vector<byte>*或realloc?
- 有些非科班生鄙视科班生，说他们只会写九九乘法表，不会用C#写私服；有些科班生鄙视非科班生，说他们基础不牢。
  I challenge both to: 把do_2_tokens和_expr改成书上那种标准的自顶向下 :-)
  就像Brian W. Kernighan在The UNIX Programming Environment里写的那样。
- cpp自顶向下+bison自底向上**混合型**实现(不支持VC6)

分类: suanpan , C/C++

Fun_with_Words
粉丝 - 10 关注 - 16
我在关注他 取消关注

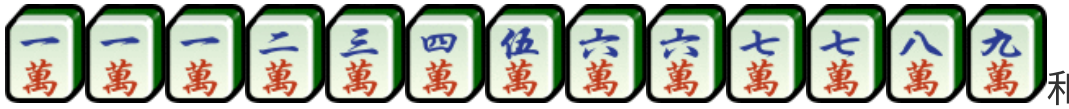
« 上一篇： WSL2, a little more
» 下一篇： antlr, C++, Demo

posted @ 2023-01-02 22:24 Fun_with_Words 阅读(133) 评论(0) 编辑 收藏 举报

· 理解 ASP.NET Core - 限流
· 一个 SpringBoot 项目能处理多少请求?

阅读排行:
· 时隔一年后，35岁生日，恰逢720
· 程序员不撰写代码注释和文档的十大理由
· 前、后端通用的可视化逻辑编排
· .Net 一套接口多实现
· 【Dotnet 工具箱】推荐一个使用 C# 开发的轻量级压测工具

历史上的今天:
2022-01-02 A Child's History of England.83
2022-01-02 A Child's History of England.82
2022-01-02 ally, alliance
2022-01-02 A Child's History of England.81
2022-01-02 A Child's History of England.80
2022-01-02 alergy
2022-01-02 A Child's History of England.79

换  和4张牌。