Программирование в Linux

Нестандартные расширения GCC/Clang

noinline и __builtin_return_address

```
5  __attribute__((noinline)) void who_call_me() {
6      void* ret_addr = __builtin_return_address(0);
7      Dl_info info;
8      dladdr(ret_addr, &info);
9
10     std::cout << "Who call me: "
11               << info.dli_fname
12               << " : "
13               << (info.dli_sname ?: "null") << std::endl;
14 }
15
16
17 inline int test() {
18     who_call_me();
19     return 5;
20 }
21
22 int main() {
23     who_call_me();
24     test();
25     []{
26         who_call_me();
27     }();
28 }
```

Для отладочных и диагностических целей можно узнавать адрес возврата из функции

```
dmis@dmis-MS-7A15:~/LinuxEgs$ g++ -g -O0 -std=c++20 -o dladdr_test dladdr_test.cpp -ldl
 -rdynamic
dmis@dmis-MS-7A15:~/LinuxEgs$ ./dladdr_test
Who call me: ./dladdr_test : main
Who call me: ./dladdr_test : _Z4testv
Who call me: ./dladdr_test : null
dmis@dmis-MS-7A15:~/LinuxEgs$ g++ -g -O3 -std=c++20 -o dladdr_test dladdr_test.cpp -ldl
 -rdynamic
dmis@dmis-MS-7A15:~/LinuxEgs$ ./dladdr_test
Who call me: ./dladdr_test : main
Who call me: ./dladdr_test : main
Who call me: ./dladdr_test : main
```

Расширения синтаксиса: case range

```
 7  enum class CharType {
 8      Digit,
 9      Alpha,
10      Other,
11  };
12
13  CharType classify (char c) {
14      switch (c) {
15      case '0' ... '9':
16          return CharType::Digit;
17      case 'a' ... 'z':
18      case 'A' ... 'Z':
19          return CharType::Alpha;
20      default:
21          return CharType::Other;
22      }
23  }
```

Расширение синтаксиса: elvis operator

```cpp
std::optional<int> next_int() {
    int x = 0;
    if (std::cin >> x) {
        return x;
    }
    return std::nullopt;
}

auto f1() {
    return next_int() ? next_int() : 0;
}

auto f2() {
    auto val = next_int();
    return val ? val : 0;
}

auto f3() {
    return [val = next_int()] {
        return val ? val : 0;
    };
}

auto f4() {
    return next_int() ?: 0;
}
```

```cpp
std::optional<int> opt1() {
    std::cout << "opt1\n";
    return std::nullopt;
}


std::optional<int> opt2() {
    std::cout << "opt2\n";
    return std::nullopt;
}

std::optional<int> opt3() {
    std::cout << "opt3\n";
    return std::nullopt;
}

int main() {
    opt1() ?: opt2() ?: opt3();
}
```

```
A  Wrap lines
ASM generation compiler returned: 0
Execution build compiler returned: 0
Program returned: 0
   opt1
   opt2
   opt3
```

Расширение синтаксиса: compound expression

```
13  void f2() {
14      const Options opt1 = []{
15          Options tmp;
16          tmp.use_a = true;
17          tmp.use_b = false;
18          return tmp;
19      }();
20  }
21
22
23  void f1() {
24      const Options opt1 = ({
25          Options tmp;
26          tmp.use_a = true;
27          tmp.use_b = false;
28          tmp;
29      });
30  }
```

Вместо IIFE (Immediately-invoked function expression). Более компактная запись.

Расширение синтаксиса: computed goto

```c
int interp_switch(unsigned char* code, int initval) {
    int pc = 0;
    int val = initval;

    while (1) {
        switch (code[pc++]) {
            case OP_HALT:
                return val;
            case OP_INC:
                val++;
                break;
            case OP_DEC:
                val--;
                break;
            case OP_MUL2:
                val *= 2;
                break;
            case OP_DIV2:
                val /= 2;
                break;
            case OP_ADD7:
                val += 7;
                break;
            case OP_NEG:
                val = -val;
                break;
            default:
                return val;
        }
    }
}
```

```c
int interp_cgoto(unsigned char* code, int initval) {
    /* The indices of labels in the dispatch_table are the relevant opcodes
    */
    static void* dispatch_table[] = {
        &&do_halt, &&do_inc, &&do_dec, &&do_mul2,
        &&do_div2, &&do_add7, &&do_neg};
    #define DISPATCH() goto *dispatch_table[code[pc++]]

    int pc = 0;
    int val = initval;

    DISPATCH();
    while (1) {
        do_halt:
            return val;
        do_inc:
            val++;
            DISPATCH();
        do_dec:
            val--;
            DISPATCH();
        do_mul2:
            val *= 2;
            DISPATCH();
        do_div2:
            val /= 2;
            DISPATCH();
        do_add7:
            val += 7;
            DISPATCH();
        do_neg:
            val = -val;
            DISPATCH();
    }
}
```

Расширение синтаксиса: designated Initializers (C only)

```
2   struct point { int x, y; };
3
4   int main() {
5
6       int a[6] = { [4] = 29, [2] = 15 };
7
8       int widths[] = { [0 ... 9] = 1, [10 ... 99] = 2, [100] = 3 };
9
10      struct point p = { .y = 3, .x = 4 };
11
12      struct point ptarray[10] = { [2].y = 5, [2].x = 6, [0].x = 0 };
13
14      return 0;
15  }
```

В C++20 поддержан ограниченный функционал