

Тема 02. Основи розробки на мові С

Давидов В.В.

Disclaimer. Recommendation

- Don't use IDE until Module4
 - Distraction from non-relevant (e.g., IDE based) issues
 - Project assembly should not be a magic for each of you
 - use vim / nano. As intermediate way – Notepad++ / Sublime / VSCode
- Don't use Windows
 - Install C/C++ developer software (and its usage) more user-friendly within Linux
 - Most CI/CD (assembly) is based on Linux hosts
- Common things:
 - Discipline yourself by using teacher's recommendation
 - Teacher don't know all possible C/C++ toolchain. In cases you manage non-recommended toolchain and get question – you may get alone with your problems

Disclaimer. Other

- Why no I/O until Module 5?
 - логічна залежність від нерозібраних тем (покажчики, функції)
 - у великих проектах простіше подивитись дані у відладоднику, а не виводити їх на екран, особливо якщо перекомпіляція триває десятки хвилин
 - віддалене налагодження (embedded devices)
 - Існують процеси (deamons/services), які мають потоки вводу/виводу
- Why don't use things we haven't learned yet
 - знання необхідно отримувати з азів. Якщо пропускати ази, то якість освіти падає.
 - Понаписуєте несистематизованого коду, а потім самі вже нічого не розумієте як воно працює чи не працює

Online-resources

- <https://labs.play-with-docker.com>
 - 4 hours free linux host with access over console
- <https://code.woboq.org/userspace/glibc/>
 - C core library sources
- <https://www.onlinegdb.com>
 - Online coding tool with ability to debug. Can be used until module 4 (don't support multifile structure)

Типи даних

Тип даних	Розмір, байт	Min	Max
char	1	-128	127
short int	2	-32768	32767
int	4	-2147483648	2147483647
long (або long int)	4/8	$-2 * 10^9$	$2 * 10^9$
long long	8	$-9 * 10^{19}$	$9 * 10^{19}$
float	4	$-1 * 10^{38}$	$1 * 10^{38}$
double	8	$-2 * 10^{308}$	$2 * 10^{308}$
long double	8/12	$-1 * 10^{4932}$	$1 * 10^{4932}$

Other:

- `sizeof` – «наше все» для визначення розміру
- `int` – відправна точка, її розмір дорівнює розмір архітектури, під яку виконується компіляція
- `short int` – в 2 рази менш за `int`
- Діапазон значень для цілочисельних типів: $\pm 2^{\text{size_in_bits} - 1}$
- Розмір типу `char` завжди 1 байт
- Зменшення точності речових даних дозволяє підвищити ємність

ТИПИ ДАНИХ

- Знакові та без знакові. Приклади беззнакових змінних - довжина, обсяг, розмір, площа. За умовчанням всі числа – знакові. Щоб явно вказати беззнаковість, необхідно при оголошенні вказати тип `unsigned`
- **Булевий тип.** У чистому Сі відсутня. Є визначення, ознайомлення з яким буде під час роботи з системними бібліотеками та функціями
- За замовчуванням, для всіх цілих констант (наприклад, 123) застосовується тип `int`, а для речових (наприклад, 3.14) – тип `double`, навіть незважаючи на те, що тип, що використовується, надмірний або, навпаки, недостатній (наприклад, для числа 5000000000). Іноді ця поведінка не влаштовує програміста, і щоб явно вказати тип цілісної константи, використовуються суфікси, що ставляться відразу після числа:
 - F (або f) – `float` (для речових). Наприклад, `3.14159f` – вже типу `float`, а не `double`.
 - U (або u) – `unsigned` (для цілих), `123U` – беззнакова константа типу `unsigned int`
 - L (або l) – `long` (для цілих та речових). Наприклад, `3.14L` – вже типу `long double`, а не просто `double`; число `123L` – типу `long`.
 - UL (або ul) – `unsigned long` (для цілих).
 - LL (або ll) – `long long int` (для цілих).

Змінні

- Змінна – ділянка пам'яті, що характеризується ім'ям, значенням, типом та адресою розташування.
- Операції, можливі над змінними:
 - оголошення;
 - float radius;
 - ініціалізація;
 - float radius;
 - radius = 24.5f;
 - поєднане оголошення та ініціалізація;
 - float radius = 24.5f;
 - використання (привласнення значення, набуття значення)
 - float radius = 24.5f;
 - float x = radius / 2;
 - radius = radius - 1;
- Константы – переменные, значения которых нельзя менять. Например, число Пі
 - #define PI 3.1415
 - const float PI = 3.14f; /* C++ специфично. Категорично не рекомендуется использовать до використання класів (например, для с11). Є лише логічними константами. Виняток – опис типу `const char *` */

Коментарі

Коментарі – пояснення до коду програми (не беруть участь у компіляції)

```
unsigned int len; /* довжина сторони квадрата */
```

- Які коментарі не варто писати:

- Очевидні коментарі (див. рисунок)
- Писати, що код складний і ніхто не розуміє, як він працює
- Писати, що код не потребує коментарів
- Писати, що нема часу писати каменти
- Чататися в коді
 - // 20/05/2020. Steve, why do we need this?
 - // 21/05/2020. John, to <bla-bla-bla>

```
1 // getting the country code
2 String country_code = get_country_code();
3
4 // if country code is US
5 if (country_code == 'US') {
6
7     // display the form input for state
8     System.out.println(form_input_state());
9 }
10
```

Оператори та їх пріоритети

- :: ,[],(),.,->, ~ (порозрядне НЕ), & (отримання адреси), * (розименування вказівника)
- Логічне «НЕ»: !
- інкремент/декремент: ++, --,
- Мультиплікативні оператори: *, /, %,
- Адитивні оператори: +, - ,
- Оператори бітового зсуву: >> , <<
- Оператори порівняння: <, >, <=, >=, ==, !=
- Порозрядні оператори: & (AND), ^ (XOR), | (OR)
- Логічне «AND» : &&
- Логічне «OR»: ||
- Тернарний оператор: (x == t) ? y: x
- Операторы присваювання: =, +=, -=, /=, *= , &=

Приклади роботи операторів

Оператори зсуву

$4 << 2 = 00000100b << 2 = 00010000b = 16;$

$5 >> 1 = 00000101b >> 1 = 00000010b = 2;$

$5 >> 3 = 00000101b >> 3 = 00000000b = 0;$

$4 << 4 = 00000100b << 2 = 01000000b = 64;$

$X << Y ==> x * (2 ^ x)$

$X >> Y ==> x / (2 ^ x)$

Булеві оператори

Table 2 – Таблиця прикладів обчислення бітових операцій

Обчислення в 10-річному форматі	Обчислення в аналогічному двійковому форматі	Результат (10-річний)	Результат (двійковий)
5 & 7	0101 & 0111	5	0101
5 & 6	0101 & 0110	4	0100
10 6	1010 0110	14	1110
7 ^ 13	0111 ^ 1101	10	1010
unsigned char a = 25;	~ 00011001	230	11100110

Table 3 – Таблиця істинності кон'юнкції (ТА), диз'юнкції (АБО), що виключає АБО (XOR):

x	y	Результат ТА	Результат АБО	Результат XOR
0 («неправда»)	0 («неправда»)	0	0	0
0 («неправда»)	1 («правда»)	0	1	1
1 («правда»)	0 («неправда»)	0	1	1
1 («правда»)	1 («правда»)	1	1	0

Перша програма

```
int main( )
```

```
{
```

визначення та ініціалізація даних;

дії над даними;

```
return 0;
```

```
}
```

Які помилки в
коді?

Перша програма

```
/* Програма обчислює об'єм циліндра з висотою 15.5 і радіусом 20 мм. */
01 int main()
02 {
    /* константа числа Пі */
03     #define PI 3.14
    /* Радіус. За умовою він у нас цілий.
       Зупинимося на відповідному типі даних */
04     #define RADIUS 20;
    /* Висота. За умовою вона у нас речова */
05     #define HEIGHT 15.5f
06     float result;
07     result = PI * RADIUS * RADIUS * HEIGHT;
08     return 0;
09 }
```

Інші помилки при написані перших програм

```
int main()
{
    int sum = 30;
    int count = 4;
    float average = sum / count;
    /* (1)
       розподіл двох цілих чисел дає ціле число. В даному випадку
       треба писати: `float average = sum / (float) count;` */

    int a;
    int b = a + 10; /* (2) змінна `a` не проініціалізована.
                      див. задачу «про буратино» */

    return 0;
}
```

Стандарти оформлення коду

- <https://www.kernel.org/doc/html/v4.10/process/coding-style.html>
- `clang-format -i` - ваш друг
- Constants - ALL_CAPICAL_CASE: MAX_ITERATIONS, PI
- Імена всіх змінних – snake_case: student_name
 - Усі абревіатури – у нижньому регистрі: export_html, read_dvd
- Назви всіх ідентифікаторів повинно однозначно визначати їх призначення (idx, index, numerator, denominator, matrix)
- Слід уникати «магічних» чисел у коді. Числа, відмінні від 0 або 1, слід повідомляти, як іменовані константи
- Якщо строкова чи числовая константа/літерал використовується більш, ніж один раз - необхідно робити з неї іменовану константу.
- Використання глобальних змінних слід уникати.
- Логічно пов'язані блоки в коді необхідно відокремлювати порожнім рядком/рядками та/або коментарями, що їх пояснюють

Отримання результатів програми

- 1. Підготовка директорії для проекту
 - mkdir -p ~/test/unit02/sample
 - cd ~/test/unit02/sample
 - git init (or clone from existing empty github/gitlab repo)
- 2. Подготовка файла сборки (Makefile, краще скопіювати з sample_project)
 - clean:
rm -rf dist
 - prep:
mkdir dist
 - compile:
clang -g main.c -o ./dist/main.bin
 - all: clean prep compile
- 3. Створення програми
 - nano main.c
 - // вводимо код «першої» програми
- 4. Компіляція
 - make all
- 5. Запуск на налагодження / перевірка результату:
 - lldb ./dist/main.bin
 - (переглядаємо код за допомогою команди list і знаходимо рядок, де результат вже є (у нашому випадку – строка з return 0))
 - b 13
 - r
 - p result
- 6. Фіксація результату
 - git add main.c Makefile
 - git commit -m "First sample"
- 7. (Опціонально заливаємо результат на віддалений сервер (якщо проект був клонований))
 - git push

Звіт з лабораторних робіт 3-6 (приклад)

- lab03.txt

Лабораторна робота №3. Розробка лінійних програм

Давидов Вячеслав Вадимович, гр. КІТ-24а

Завдання: визначити об'єм конуса.

Основна частина:

- опис роботи основної функції: об'єм конуса визначається за формулою: $V = 1/3 * \pi * r^2 * h$, де π - константа, h - висота конуса, r - радіус основи конуса.
- перелік вхідних даних:
 - h - висота конуса, позитивне дійсне число (float);
 - r - радіус основи конуса, позитивне дійсне число (float).
- перелік констант:
 - π - математична константа. Дорівнює значенню 3.14159.
- дослідження результатів роботи програми:
 - V - об'єм конуса. Так як всі операємі числа є дійсними, то й результируча змінна є дійсною. Також, слід визначити, що об'єм конуса не може бути негативним. Тому, її тип - float
 - при значенні $h=2.5$ та $r=12$, об'єм конуса повинен складати: $1/3 * 3.14159 * 12 * 12 * 2.5 = 376.9908$
 - для підтвердження коректності роботи програми, зупинено відлагодник на строці з "return 0" та введено команду "print V". Після вводу команди отримали наступне:

```
(lldb) print V  
(float) $1 = 376.990814
```

Як бачимо, результат майже співпав. Похибка 0.000014 є дозволеною при виконанні операцій з плаваючою крапкою. Подібність результатів говорить про те, що програма працює коректно.

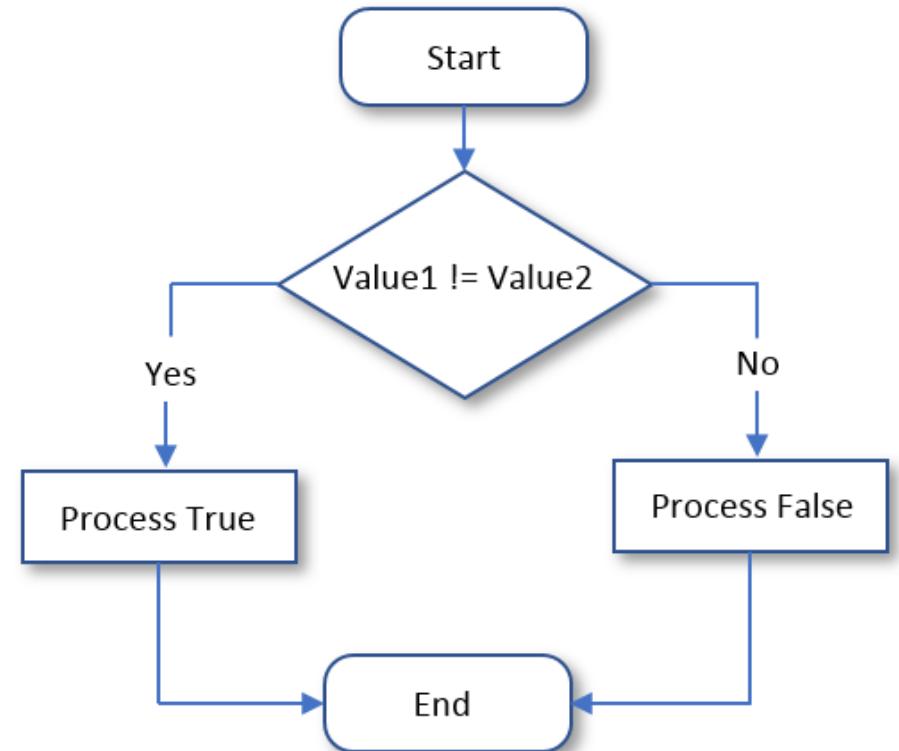
Структура проекту лабораторної роботи:

```
└── lab03  
    ├── README.txt  
    └── Makefile  
    └── src  
        └── main.c
```

Висновки: при виконанні лабораторної роботи буди набуті практичні навички створення лінійних програм на мові С, зокрема: визначати типи даних, обчислювати математичні вирази, емулювати операцію зведення до ступеню через операцію множення.

Умовні оператори

- (не)Виконання дії за певних умов
- Приклад:
 - Передмова: ви граєте в гру на телефоні
 - Перед-дія: ви звертаєте увагу на процент заряду вашого смартфону
 - Умова: якщо залишилося меньш ніж 10% заряду – ви підключаете телефон до зарядного пристрою
 - Пост-дія: ви продовжуєте грати у гру незважаючи на виконання умови.



Умовні оператори. `If` how to

```
// попередні дії
if (умова) {
    // дії при виконанні умови
} else {
    // дії при невиконанні умови
}
// пост-дії

gaming;
accum_percentage = get_current_percentage;
<bool> need_charge = accum_percentage < 10;
if (need_charge) {
    connect_charger;
}
continue_gaming;
```

`if` samples

- Якщо на вулиці йде дощ – треба взяти парасольку:
 - `if (raining) take_umbrella;`
- Якщо на годиннику якнайменш 6 ранку – пора прокидатись:
 - `if (time >= 6) wake_up;`
- Якщо нам не подобається трек – переключаємо на наступний:
 - `if (!like_song) next_song;`

Оператори порівняння:

`<, >, <=, >=, ==, !=`

Оператори скорочення:

- (йде_дощ == так) \rightarrow (йде_дощ)
- (подобається == ні)
 - \rightarrow (не подобається)
 - \rightarrow (!подобається)

Інше:

- Так / ні – булеві вирази, мають значення
 - true / false
 - 1 / 0

`if` sample. abs()

```
int main()
{
#define VALUE -34
    int result = VALUE;      /* we can't modify const value */
    if (VALUE < 0) {
        result = -VALUE; /* identical to: `result = 0 - VALUE;` */
    }
    return 0;
}
```

Composite `if` condition

- `if (temperature > 25 && !clouds) go_swimming;`
- `If (has_money && want_eat) buy_doner;`
- `if (green_trafficlight || (has_crosswalk && !cars_around)) crossing_road`
- Використання композитних умов на прямій натуральних чисел
 - $x \geq 5 \&\& x \leq 10$ // $x \in [5, 10]$
 - $x \leq 5 \&\& x \geq 10$ // N/A
 - $x \leq 5 \mid\mid x \geq 10$ // $x \in [-\infty, 5] \cup [10, +\infty]$

If-else

```
int main()
{
    #define VALUE -34
    int result;
    if (VALUE < 0) {
        result = -VALUE; /* identical to: `result = 0 - VALUE;` */
    } else {
        result = VALUE;
    }
    return 0;
}
```

If – else if

```
/* choose direction */  
if (direction == LEFT) {  
    /* Loose the horse */  
} else {  
    if (direction == RIGHT) {  
        /* Be dead */  
    } else {  
        if (direction == FORWARD) {  
            /* be rich */  
        } else {  
            /* no condition. Back to normal life */  
        }  
    }  
}
```



If – else if

```
/* choose direction */  
if (direction == LEFT) {  
    /* Loose the horse */  
} else if (direction == RIGHT) {  
    /* Be dead */  
} else if (direction == FORWARD) {  
    /* be rich */  
} else {  
    /* no condition. Back to normal life */  
}
```

"... if you need more than 3 levels of indentation, you're screwed anyway, and should fix your program." - Linus Torvalds, *Linux kernel coding style*

P.S. AFAIK, viable for C and go

If – else if. Sample 2

```
int value = -5;  
if (value > 0) {  
    /* Positive */  
} else if (value < 0) {  
    /* Negative */  
} else {  
    /* 0 */  
}
```

Ternary operator

```
bool isPraepostor = false; /* староста? */
int studentScholarship;
if (isPraepostor) {
    studentScholarship = 1500;
} else {
    studentScholarship = 1000;
}
→
bool isPraepostor = false; /* староста? */
int studentScholarship = (isPraepostor ? 1500 : 1000);
```

Switch case

```
/* choose direction */  
switch(direction) {  
    case LEFT: /* Loose the horse */; break;  
    case RIGHT: /* Be dead */; break;  
    case FORWARD: /* be rich */; break;  
    default : standby();  
}
```

Стандарти оформлення коду

- Префікс *is* слід використовувати тільки для булевих (логічних) змінних і методів, наприклад: *is_set()*, *is_visible()*, *is_finished*, *is_found*, *is_open*.
- У деяких ситуаціях префікс *is* краще замінити на іншій: *has*, *can* або *should*:
 - *bool has_license;*
 - *bool can_evaluate;*
 - *bool should_sort;*
- Не можна давати булевим (логічним) змінним імена, що містять заперечення:
 - *bool is_error;*
 - ~~is_no_error~~ або ~~no_error~~
 - *bool is_found;*
 - ~~is_not_found~~ або ~~not_found~~

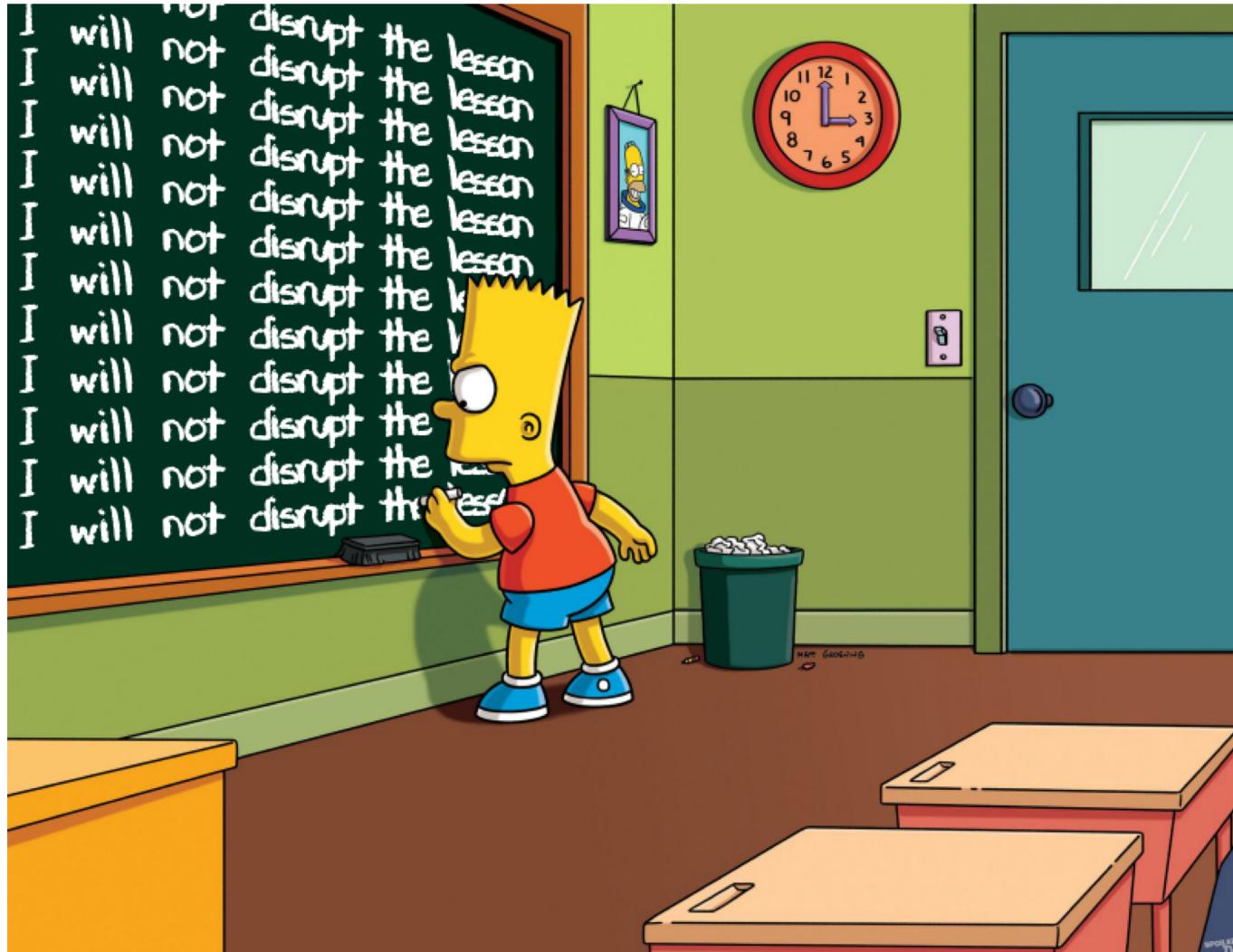
Стандарти оформлення коду. Умовні оператори

```
if (condition) {  
    //...  
} else if (condition) {  
    //...  
} else {  
    //...  
}
```

Складних умовних виразів слід уникати. Краще замість цього використовувати булеві змінні:

```
bool is_finished = (element_no < 0) || (element_no > max_element);  
bool is_repeated_entry = element_no == last_element;  
if (is_finished || is_repeated_entry) {  
    //...  
}
```

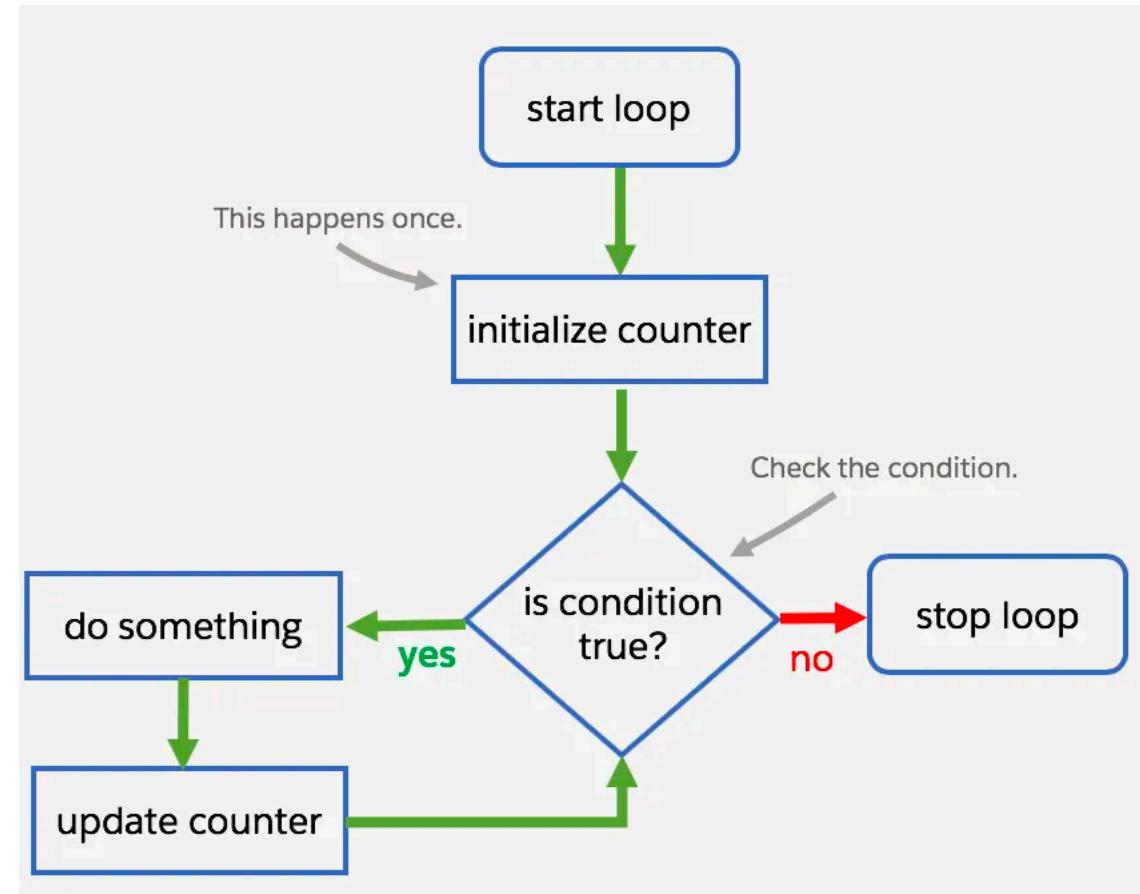

Оператори Циклу (loops)



Цикл – форма організації роботи програми, при якій одна і та ж послідовність дій виконується доки діє/валідно задана умова.

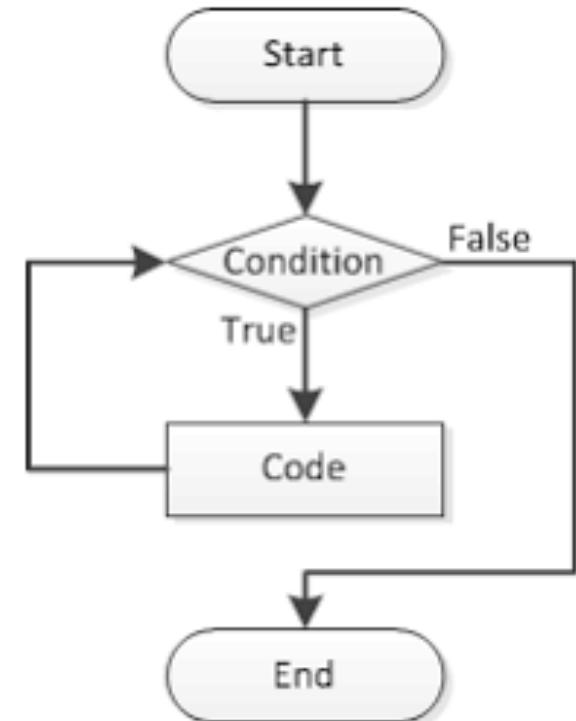
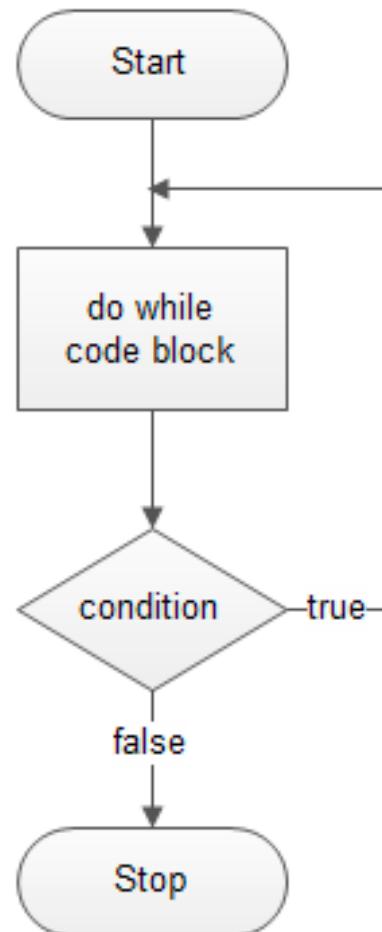
Оператори циклу

- For:
 - Універсальний. Застосовується, коли є чітко визначені межі
 - `for (<init>; <condition>; <state_update>) { <action> };`
 - `for (int i = 0; i < N; i++) { op; }`
 - `for (int i = 0; i < N; i+=10) { op; }`



Оператори циклу

- **do while**
 - Дія повинна виконатись хоча б раз (або спочатку зробив – потім подумав)
 - `do { check_water; } while (!boils);`
- **while do**
 - `while (has_catridge) { do_shot; }`
 - `while (!at_door) { do_step; }`



Оператори циклу. Обмеження

Працював в одному банку програміст, все було чудово, але одного чудового дня зник. День на роботі нема, два... Директор банку дізнався - наказав відшукати. Поїхали до програміста додому. Двері на дзвінки не відчиняють, але за дверима чути пlesкіт води. Вирішили ламати двері. Заходять у квартиру – програміст у дуже жалюгідному вигляді сидить у ванній: синій, на голові майже немає волосся, в руці стискає пляшку від шампуню. Вирвали у нього пляшку - на ній Інструкція із застосування шампуню:

- Намочити голову,
- Видавити невелику кількість шампуню на руку,
- Розтерти шампунь по волоссю,
- Змити водою,
- Повторити.

Wrong:

- while (has_catridge) { do_shot; }
- while (has_catridge) { do_shot; bullets--; }

Good:

- while (has_catridge) {
 - do_shot;
 - Bullets_count--;
 - has_catridge = !(bullets_count == 0)
- }

Оператори циклу. Continue. Break

```
int main () {
    /* find count of mod3 and mod5 in 0..VALUE range; skip mod7 */
    const int HI_VALUE = 100;
    unsigned int mod3count = 0;
    unsigned int mod5count = 0;
    for (int i = 0; i < HI_VALUE; i++) {
        if (i % 7 == 0) {    continue;    }
        if (i % 3 == 0) {    mod3count++;    }
        if (i % 5 == 0) {
            mod5count++;
        }
    }
    return 0;
}
```

Оператори циклу. Continue. Break

```
int main () {
    /* find count of mod3 and mod5 until first mod7 occurred */
    const int HI_VALUE = 100;
    unsigned int mod3count = 0;
    unsigned int mod5count = 0;
    for (int i = 0; i < HI_VALUE; i++) {
        if (i % 7 == 0) {      break;      }
        if (i % 3 == 0) {      mod3count++;    }
        if (i % 5 == 0) {
            mod5count++;
        }
    }
    return 0;
}
```

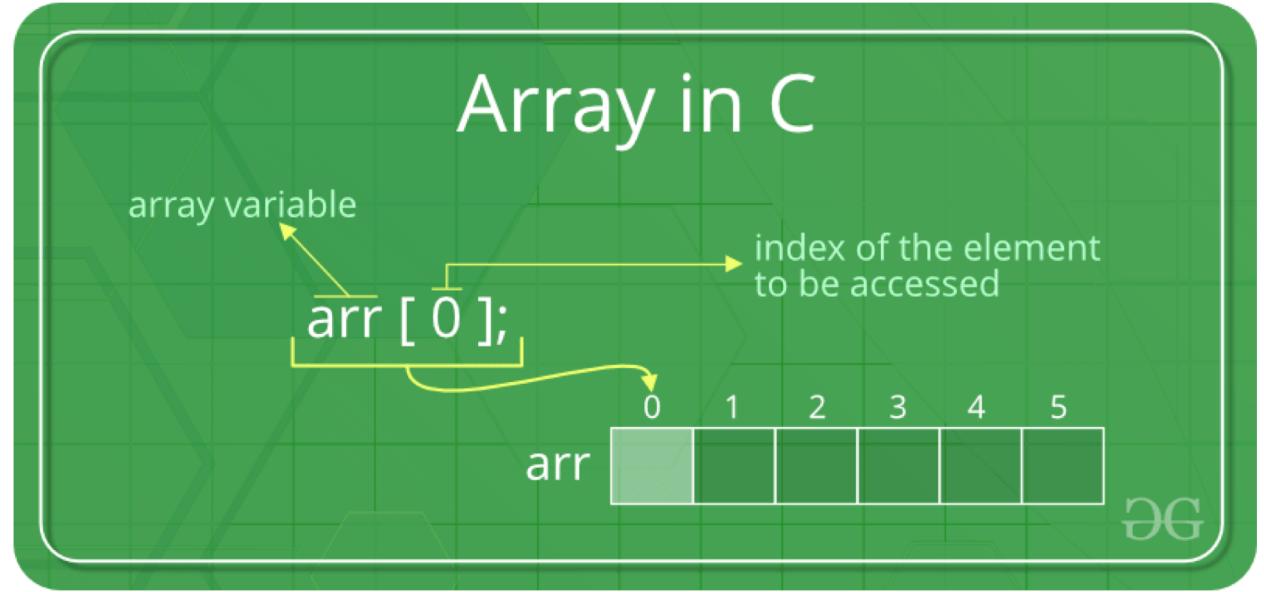
Стандарти оформлення коду. Цикли

- Слід уникати
 - цикли *do-while*
 - Нескінченні цикли: *while (true) { (дія) }*
 - Як з них можна вийти?
- Слід використовувати пропуски та відступи для покращення читаємості
 - “Добрі” Приклади:
 - `while (true) {`
 - `for (i = 0; i < 10; i++) {`
 - Погані приклади:
 - – `while(true){`
 - `for(i=0;i<10;i++){`

Arrays

Масив – набір однотипних даних, які розміщаються в пам'яті один за одним (без перепусток).

Основне (нематичне) завдання масиву – організація пошуку елемента/ів за критерієм



Де використовуються масиви:

- Статистична обробка
- Вектори, матриці (GameDev, MachineLearning)
- Агрегація даних

Why do we need arrays?

Before:

```
float best1;  
float best2;  
float best3;  
float best4;  
float best5;  
float best6;  
float best7;  
float best8;  
float best9;  
boolean level1;  
boolean level2;  
boolean level3;  
boolean level4;  
boolean level5;  
boolean level6;  
boolean level7;  
boolean level8;  
boolean level9;
```

After:

```
float[] best_results;  
boolean[] levels;
```

Pros:

- Less code
- Extendable (can add more levels with less pain)

Arrays. Samples

- Array is:
 - Students within group
 - Country citizen
 - Grades per semester
 - Car wheels
- Array is not:
 - Речі в жіночої сумці (фкщо усі речі різні і не пов'язані між собою)
 - { 123, 'c', false } (python turple)
 - координати точки
 - Набор из трав и деревьев
 - Але Масив, ящо вони усі «рослини» з узагальненими характеристиками
 - Набір з теслярів та викладчів
 - Але Масив, ящо вони усі «співробітники» з узагальненими характеристиками

Масиви. Ініціалізація та використання

- <type> <variable_name> = value; /* як ми використовуємо звичайні змінні */
- <type> <variable_name>[<size>] = { <value1>, <value2> ... };
 - size – константне значення або вираз
- int values[10];
 - TODO: завдання про буратіно
- int values[3] = {1, 2, 3};
- int another_values[3] = {1}; /* 1, 0, 0 */
- int x = values[0]; /* нумерація починається з 0 */
- values[2] = 123; /* values = 1, 2, 123; останній ел-т → [size-1] */
- values[1]++; /* values = 1, 3, 123 */
- values[10] = 123; /* ??? */
- int unknown_values[] = { 1, 2, 3}; /* створився масив із 3х ел-тов. Обов'язкова inline ініціалізація */

Двовимірні масиви

- У мовах програмування багатовимірні масиви – масиви масивів на 1 меншого розміру.
 - 2D масив – масив 1D масивів (векторів)
 - 3D масив – масив 2D масивів = масив масивів масивів
- Приклад 2D масиву – матриця, 2D координатна сітка
- Приклад 3D масив - 3D координатна сітка, кубик-Рубіка

```
int matrix_plain[2*3] = { a11, a12, a13 , a21, a22, a23 } ;
```

$$: \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix}$$

```
int m1[3] = {a11, a12, a13 }; int m2[3] = {a21, a22, a23 };  
int matrix[2][3] = {m1, m2};
```

```
int matrix[2][3] = { { a11, a12, a13 },  
                     { a21, a22, a23 } } ;  
matrix[0][2] = 3; /* a13 = 3 */  
matrix[0][3] = 4; /* [a14 =] a21 = 4 */
```

Масиви та цикли. Заповнення масиву

```
int main() {  
    #define N 5  
    int values[N];  
    for (int i = 0; i < N; i++) {  
        values[i] = i + 1;  
    }  
    return 0;  
}
```

```
int main() {  
    #define N 5  
    #define M 6  
    int values[N][M];  
    for (int i = 0; i < N; i++) {  
        for (int j = 0; j < M; j++) {  
            values[i][j] = i * N + j + 1;  
        }  
    }  
    return 0;  
}
```

Масиви та цикли. Пошук min/max

```
int main () {  
    #define N 5  
    int values[N] = {1, 4, -4, 6, 2};  
    int min = MAX_INT;  
    int max = MIN_INT;  
    for (int i = 0; i < N; i++) {  
        if (min > values[i]) { min = values[i]; }  
        /* no else/ else if !!! */  
        if (max < values[i]) { max = values[i]; }  
    }  
    return 0;  
}
```

```
int main () {  
    #define N 5  
    int values[N] = {1, 4, -4, 6, 2};  
    int min = values[0];  
    int max = values[0];  
    for (int i = 1; i < N; i++) {  
        if (min > values[i]) { min = values[i]; }  
        if (max < values[i]) { max = values[i]; }  
    }  
    return 0;  
}
```

Масиви. Сортування «бульбашкою»

```
int main() {
#define N 5
    int values[N] = {1, 4, 2, 6, -1};
    for (int i = 0; i < N - 1; i++) {
        for (int j = 0; j < N - i - 2; j++) {
            if (values[j] > values[j+1]) { /* sign = sort order */
                int temp = values[j];
                values[j] = values[j + 1];
                values[j + 1] = temp;
            }
        }
    }
    return 0;
}
```

{1, 4, 2, 6, -1};
i = 0; j = 3: {1, 2, 4, -1, **6**}
i = 1; j = 2: {1, 2, -1, **4, 6**}
i = 2; j = 1: {1, -1, **2, 4, 6**}
i = 3; j = 0: {-1, **1, 2, 4, 6**}

int min = values[0];
int max = values[N - 1];

Масиви. Знаходження дисперсії вибірки

```
int main () {  
    #define N 5  
    int values[N] = {1, 4, -4, 6, 2};  
    int sum = 0;  
    for (int i = 0; i < N; i++) {  
        sum += values[i];  
    }  
    float average = sum / (float) N;  
    float dispersion = 0;  
    for (int i = 0; i < N; i++) {  
        /* or we can extract (values[i] - average) to variable */  
        dispersion += (values[i] - average) * (values[i] - average);  
    }  
    dispersion /= (N - 1);  
    return 0;  
}
```

$$S^2 = \frac{\sum (x_i - \bar{x})^2}{(n-1)}$$

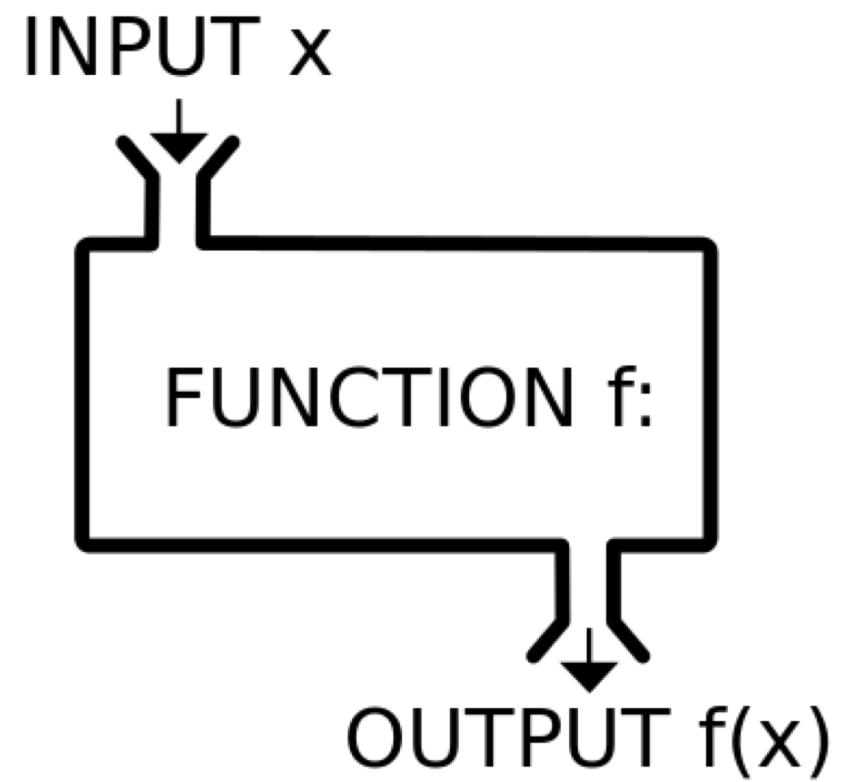
Character arrays. Strings (NTCS)

- `char symbol = 'X';`
- `char string1[5] = { 'H', 'e', 'l', 'l', 'o' };`
- `char str2[] = "Hello, world"; /* 13 characters */`

Функції

Функція — в математиці відповідність між елементами двох множин, встановлена за таким правилом, що кожному елементу першої множини відповідає один і тільки один елемент другої множини.

Функція в програмуванні (thecode.media) - це міні-програма всередині вашої основної програми, яка робить якусь одну зрозумілу річ. Ви якось описуєте, що це за річ, а потім посилаєтесь на цей опис.



Функції. Приклади

- Знайти мінімум/максимум з вибірки.
 - Вхід: масив та його розмір
 - Вихід: число
- !!!№1 Отримання випадкового числа
 - Вхід: пустий
 - Вихід: випадкове число
- !!!№2 Вивід на екран (будь-що, наприклад масив)
 - Вхід: масив та його розмір
 - Вихід: пустий/відсутній
- Заповнення журналу пропусків
 - Вхід: незаповнений журнал
 - Вихід: заповнений журнал

Функції. Класифікація

- За типом підключення:
 - Користувача
 - Бібліотечні
 - Стандартні (напр., printf)
 - Користувача (що створені вами або іншими розробниками, див. опцію `-I` компілятора)
 - Системні:
 - sizeof
- За типом повернення значень
 - Повертають результат (застосовано до дій такий як «дай», « знайди», «є / маєш ?»)
 - Не повертують результат («процедури») (застосовано до дій «зроби»)

Функції. Використання

```
<return_type> <name> (<arguments>) {  
    <body>  
    return <return_type_variable>;  
}  
  
int getSum(int a, int b) { /* wrong getSum(int a, b) */  
    int result = a + b;  
    return result; /* = return a + b; */  
}  
  
void processMe(int argument) {  
    /* some action */  
    /* return; */  
}  
  
int rand() {  
    return some_value;  
}
```

```
int main() {  
    int var1 = 3;  
    int var2 = 5;  
    int c = getSum(4, 5);  
    int d = getSum(var1, var2);  
  
    int e = rand();  
  
    processMe(var1);  
    return 0;  
}
```

ФУНКЦІЇ, ЩО НЕ ПРИЙМАЮТЬ АРГУМЕНТИ

```
void func1() { }
```

```
void func2(void) { }
```

```
int main() {
    func1();
    func1(5); /* warning: too many arguments in call to 'func1' */
    func1(3, 5); /* warning: too many arguments in call to 'func1' */
    func2();
    func2(5); /* error: too many arguments to function call, expected 0, have 1 */
    return 0;
}
```

ФУНКЦІЇ. Попереднє оголошення

```
/* 1. how we used to  
use */  
  
void func1(args) { .. }  
void func2(args) { .. }  
  
(line 100500) int main() {  
    func1(..);  
    func2(..);  
    ...  
}  
...
```

```
/* 2. how we want to use */  
  
int main() {  
    func1(..);  
    func2(..);  
    ...  
}  
  
void func1(args) { .. }  
void func2(args) { .. }
```

```
/* 3. how we should use */  
  
void func1(args);  
void func2(args);  
  
int main() {  
    func1(..);  
    func2(..);  
    ...  
}  
  
void func1(args) { .. }  
void func2(args) { .. }
```

Бібліотечні (системні) функції

- `#include <filename.h>`
 - h - header files
 - contains only prototypes
 - and constants
 - .h files are **not library!**
 - library -> -l gcc option
- `#include "filename.h"`
- `math.h`
 - `float cosf(float);`
 - `double cos(double);`
 - `long double cosl(long double);`
 - `double pow(double, double)`
 - `double sqrt(double)`
 - `double ceil(double)`
 - `double floor(double)`
 - `double round(double)`
- `stdbool.h`
 - `bool flag = true; /* or false */`
 - `_Bool flag = 1; /* or 0 */`
- `stdint.h`
 - `INT8_MAX , INT32_MAX , INT16_MIN`
 - `int32_t , int8_t, uint8_t, size_t`
- `stdlib.h`
 - `system(char[])`
 - `double strtod(const char *, char **)`
 - don't use atoi
 - `void exit(int)`
 - `int rand();`
 - `x = a + rand()%(b-a); /* x = rand of [a .. b) */`
 - `stdlib.h + time.h = `rand` randomizer`
 - `srand(time(NULL))`

Функції. Передача даних «за значенням» і не лише

```
void weird_func(int a, int array[]) {  
    a++;          /* 11 */  
    array[1] = a; /* 0, 11, 0 */  
}  
  
int main() {  
    int a = 10;  
    int b[3] = { 0 };  
    weird_func(a, b); /* weird_func(10,  
{0,0,0}); */  
  
    /* a, b - ?? */  
}
```

- Don't return arrays via `return`
- For arrays pass size
- Function can return only 1 item, otherwise you have problems
- void sort(int array[], int size);

Передача багато-вимірних масивів до функції

```
void set_value(int array[], int size_x, int x, int y, int value) {  
    // see: https://en.wikipedia.org/wiki/Row- and column-major\_order  
    array[size_x * y + x] = value;  
}  
  
int main(void) {  
    int x[3][3] = {  
        {1, 2, 3},  
        {4, 5, 6},  
        {7, 8, 9}  
    };  
    set_value((int *)x, 3, 2, 1, 777);  
  
    return 0;  
}
```

Note:

- (int *) is not required, but compiler would make warnings otherwise

/*

{1, 2, 3},
{4, 5, 777},
{7, 8, 9}

*/

Note2: no way for passing multi-dimention arrays
of strings or complex data without hardcoded
(and pointers) in C

ФУНКЦІЇ. Варіативні функції

```
#include <stdarg.h>
double average(int count, ...) {
    va_list ap;
    int j;
    double sum = 0;
    /* Requires the last fixed parameter (to get the address) */
    va_start(ap, count);
    for (j = 0; j < count; j++) {
        /* Increments ap to the next argument. */
        sum += va_arg(ap, int);
    }
    va_end(ap);
    return sum / count;
}

int main(int argc, char const *argv[]) {
    float avg = average(3, 1, 2, 3);
    return 0;
}
```

- https://en.wikipedia.org/wiki/Variadic_function
- Повинен бути або ознака кінця масива, або їх кількість
- Expert level - How va is represented:
 - `typedef unsigned char *va_list;`
 - `#define va_start(list, param) (list = (((va_list)¶m) + sizeof(param)))`
 - `#define va_arg(list, type) (*(type *)((list += sizeof(type)) - sizeof(type)))`

Функція main()

- `return 0` - повернення **1** байту – статусу завершення (код помилки) **програми**
 - Прийнято, що код повернення "0" успішно. 0 – ідентичний false, таким чином виходить що «bool has_errors = program_main(args)»
 - Інші коди символізують якусь певну помилку
 - int vs 1 byte -> padding
 - int main() – даніна минулому. У деяких компіляторах можна використовувати `void main()` (**но в л/р забороняється!**)
 - echo \$?
 - operation && on_success || on_failure
- Не використовуйте return з функції main для повернення даних!!!
- main(int argc, char** argv)
 - ./main.bin
 - argc = 1
 - argv[0] = "/Users/davs/CLionProjects/test/cmake-build-debug/test"
 - ./main.bin param1
 - argc = 2
 - argv[1] = "param1"

Приклади кодів повернення в утилітах

Linux:

- 1 - Catchall for general errors
- 126 - Command invoked cannot execute
- 127 - “command not found”
- 128 - Invalid argument to exit
- 128+n - Fatal error signal “n”
- 130 - Script terminated by Control-C
- 255 - Exit status out of range

Функції. Рекурсивна обробка

- Рекурсія – зло
 - (с) Щоб зрозуміти, що таке рекурсія, треба зрозуміти, що таке рекурсія
 - 90%+ всіх завдань можна вирішити ітеративно (циклами), навіщо ускладнювати собі життя?
 - код працює повільніше, додаткове навантаження на стек (що може привести до його переповнення)
 - Потрібно контролювати критерій закінчення рекурсії
- Рекурсія – це не так уж і погано
 - код математично/алгоритмічно красивіший. Усі функціональні мови програмування «заточені» під роботу з рекурсією
 - є ряд завдань, які майже не можливо виконати по іншому, наприклад, при роботі з «деревами» - показ файлів і каталогів углиб рекурсивно (наприклад, як утиліта `tree`)

Обчислення факторіалу:

```
unsigned long long int factorial(unsigned int i) {  
    if(i <= 1) {  
        return 1;  
    }  
    return i * factorial(i - 1);  
}
```

$$\begin{aligned} 5! &= \text{factorial}(5) = \\ &= 5 * \text{factorial}(4) = 5 * (4 * \text{factorial}(3)) \\ &= 5 * (4 * (3 * \text{factorial}(2))) = \\ &= 5 * (4 * (3 * (2 * \text{factorial}(1)))) = \\ &= 5 * (4 * (3 * (2 * 1))) = 120 \end{aligned}$$

Get execution time

```
#include <time.h>  
...  
clock_t start = clock(); /* `clock_t` - e.g. unsigned long */  
/* TODO: heavy operation here */  
clock_t end = clock();  
double time_used = ((double) (end - start)) / CLOCKS_PER_SEC;
```

Why return array (even of primitives) is bad

- Area of visibility and amount of data ambiguous

```
int main() {  
    int a = 5;  
    if (a > 0) {  
        int b = a + 1; // `a` is accessible  
        // do smth with b  
    }  
    // b is inaccessible at this point  
    // ...  
}
```

```
int * func(int[] array, int N) {  
    // y is alive so far, so array is viable  
    int x[] = { array[0], array[1] }  
    return x;  
}  
  
int main() {  
    int y[] = {1, 2, 3};  
    int z[] = func(y, 3);  
    // 1. x is ruined. Memory z is used can be  
    // overridden by something else  
    // 2. how many elements would z contain?  
    // We can just guess  
}
```

Note: tss.. The main problem is with stack .. See Unit05