**Лекція 4. Основні поняття алгоритмічної мови, символи, елементарні конструкції.**

1. Основні поняття
2. Нотація Бекуса – Наура
3. Синтаксичні діаграми
4. Алфавіт мови програмування
5. Лексеми
6. Елементи даних
7. Операції.
8. Вирази
9. Обробка виразів. Використання функцій.
10. Оператори.

**Основні поняття**

Звичайна розмовна мова складається з чотирьох основних елементів: символів, слів, словосполучень і речень. Алгоритмічна мова містить подібні елементи, тільки слова називають елементарними конструкціями, словосполучення - виразами, речення - операторами.

Символи мов – це основні неподільні знаки, якими пишуться всі тексти певною мовою. Елементарні конструкції (лексеми) – це мінімальні одиниці мови, що мають самостійний зміст. Вони утворяться з основних символів мови. Вираз в алгоритмічній мові складається з елементарних конструкцій і символів, він задає правило обчислення деякого значення. Оператор задає повний опис деякої дії, що необхідно виконати.

Символи, елементарні конструкції, вирази і оператори складають ієрархічну структуру, оскільки елементарні конструкції утворюються з послідовності символів, вирази – це послідовність елементарних конструкцій і символів, а оператор – послідовність виразів, елементарних конструкцій і символів.

Мову програмування, як і будь-яку формальну мову, визначає три її складові: алфавіт, синтаксис (граматика) і семантика, що задають зовнішній вигляд програми і дії, які виконує комп'ютер під її управлінням.

***Алфавіт мови програмування*** (її термінальний словник) - це фіксований набір символів, які дозволяється використовувати для утворення нетермінальних конструкцій мови (її нетерміналів).

Синтаксичні визначення (правила) встановлюють правила побудови допустимих конструкцій мови із символів її алфавіту.

***Семантика*** визначає зміст і правила використання мовних конструкцій, для яких були дані синтаксичні визначення. Семантичні правила пояснюють, яке смислове значення має кожна мовна конструкція і які дії повинен виконати комп'ютер під час їх виконання (виконання команди програми).

Порушення форми запису програми приводить до її нерозуміння ПК і видачі повідомлення про синтаксичну помилку, а використання правильно написаних команд, але таких, що не відповідають необхідній послідовності дій, приводить до семантичних помилок (їх ще називають логічними помилками, а також помилками часу виконання).

Описом мови є опис чотирьох названих вище елементів:

1. опис символів полягає в перерахуванні припустимих символів мови;
2. під описом елементарних конструкцій розуміють правила їхнього утворення;
3. опис виразів – це правила утворення будь-яких виразів, що мають зміст у даній мові;
4. опис операторів складається з розгляду усіх типів операторів, припустимих у мові.

При описі мови програмування і при побудові алгоритмів використовуються певні *поняття* мови. Такий підхід, до речі, застосовується і в шкільних курсах, зокрема, геометрії. Так, щоб визначити, що таке «прямокутний трикутник», треба ввести поняття «трикутник», «кут», «прямий кут», тощо.

В будь-якій мові програмування можна виділити наступні базові поняття:

- алфавіт,

- лексеми,

- елементи даних,

- операції,

- вирази,

- оператори.

**Нотація Бекуса - Наура**

**Нота́ція Бе́куса — Нау́ра** (англ. *Backus-Naur form*, *BNF -* БНФ) — це спосіб запису правил формою опису формальної мови. Саме її типово використовують для запису правил мов програмування та протоколів комунікації. У 50-х роках минулого сторіччя Джон Бекус створив цю нотацію розробляючи мову ALGOL. На першому Всесвітньому Комп'ютерному Конгресі, що відбувся у [Парижі](https://uk.wikipedia.org/wiki/%D0%9F%D0%B0%D1%80%D0%B8%D0%B6) [1959](https://uk.wikipedia.org/wiki/1959)-го він зробив доповідь на тему «Синтаксис та семантика пропонованої першої міжнародної алгебраїчної мови». Пізніше Пітер Наур спростив її та (за порадою Дональда Кнута) додав до назви своє ім'я.

БНФ визначає скінченну кількість символів (нетерміналів). Крім того, вона визначає правила заміни символу на якусь послідовність букв (терміналів) і символів. Процес отримання ланцюжка букв можна визначити поетапно: спочатку є один символ (символи зазвичай знаходяться у кутових дужках, а їх назва не несе жодної інформації). Потім цей символ замінюється на деяку послідовність букв і символів, відповідно до одного з правил. Потім процес повторюється (на кожному кроці один із символів замінюється на послідовність, згідно з правилом). Зрештою , виходить ланцюжок , що складається з букв і не містить символів. Це означає , що отриманий ланцюжок може бути виведений з початкового символу .

Нотація БНФ є набором «продукцій», кожна з яких відповідає зразку:

**<символ> ::= <вираз, що містить символи>**

де <вираз, що містить символи> це послідовність символів або послідовності символів, розділених вертикальною рискою |, що повністю перелічують можливий вибір символ з лівої частини формули.

Далі,

* < — лівий обмежувач виразу
* > — правий обмежувач виразу
* ::= — *визначене як*
* | — *або*

Ці чотири символи є символами мета-мови, вони не визначені у мові, котру описують. Решта описаних символів належать до «абетки» описуваної мови.

Приклад 1. Один зі способів означити натуральні числа за допомогою БНФ.

<нуль> ::= 0

<ненульова цифра> ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<цифра> ::= <нуль> | <ненульова цифра>

<послідовність цифр> ::= <нуль> | <ненульова цифра> | <цифра><послідовність цифр>

<натуральне число> ::= <цифра> | <ненульова цифра><послідовність цифр>

**Синтаксичні діаграми**

Ще одним способом опису мови є синтаксичні діаграми. На наданому нижче рисунку а) – оголошення змінної, б) – оголошення функції.



Формальній мові семантика необхідна тоді, коли будується реалізація мови (транслятор). Саме тут зазвичай використовують БНФ та синтаксичні діаграми.

**Алфавіт мови** **програмування**.

Зазвичай, алфавіт мови програмування (її термінальний словник) являє собою підмножину набору символів коду ASCІІ (Unicod). Так алфавіт мови С/С++ складають:

- рядкові і прописні літери латинського алфавіту, арабські десяткові цифри (0 - 9), а також символ підкреслення (“\_”), що служать для завдання різних ідентифікаторів;

- спеціальні символи, що використовуються для конструювання знаків операцій, виразів, коментарів, а також як синтаксичні роздільники;

- символ пробілу (код *32*), символи табуляції, що використовуються як роздільники;

- керуючі символи (коди 0 - 31), що служать для завдання різних специфічних дій.

Так термінальний словник мови С/С++ може бути описаний за допомогою наступних правил граматики:

<символ> ::= <літерал> | <цифра> | <спеціальні символи>

<спеціальні символи> = <+ | - | \* | …>.

Більш детально спеціальні символи розглянемо в наступних лекціях при вивченні мови С++.

**Лексеми**.

Із символів алфавіту будуються *лексеми* - мінімальні значимі одиниці мови, що мають певний зміст (її нетермінальний словник). Множина всіх допустимих лексем називається *словником* мови програмування (нетермінальним словник мови).

Розрізняють наступні основні види лексем.

*Ідентифікатори*. Це слова, що служать для завдання імен програмних об'єктів (імен змінних, констант, підпрограм, програм тощо).

Правила побудови ідентифікаторів для різних мов програмування можуть бути різними. Так, з точки зору синтаксису багатьох мови, зокрема, C/C++, *ідентифікатор* – це довільна послідовність літер латинського алфавіту, цифр і знаку підкреслення, що починається з літери або з символу підкреслення і не містить пробілів:

<ідентифікатор>:: = <літера> | {літера | цифра}.

Символ підкреслення можна використовувати для поліпшення читабельності імені змінної, наприклад first\_name.

У мовах програмування має місце різниця у сприйнятті рядкових і прописних літер алфавіту. Так у мові C/C++ прописні і рядкові букви сприймаються як різні символи, тобто myvar і MyVar – різні імена, тоді як у Рascal ідентифікатори байдужні до регістра клавіатури.

У будь-якому випадку рекомендується вибирати змістовні ідентифікатори, використовуючи для наочного їх представлення прописні і рядкові літери. Наприклад, замість ідентифікатора nomerotdela, краще написати NomerOtdela, виділивши прописними буквами кожну із двох змістовних частин.

*Зарезервовані (ключові)* *слова*. Це слова, що мають у мові програмування певне призначення, яке не може змінюватися. Зарезервовані слова використовуються для позначення алгоритмічних конструкцій, розділів програми тощо.

Приклади список ключових слів мови С/С++:

Таблиця 1. Список ключових слів мови С++

****

*Літерали* (рядки символів). Це певним чином оформлена послідовність символів коду ASCІІ (Unicod).

Так у мові С/С++ літерали – це рядки символів, взяті у подвійні лапки, наприклад, *”Приклад”*, у мові Рascal– вони беруться в апострофи, наприклад, *'Приклад'.* Рядок символів, який нічого не містить між обмежувачами, є порожнім рядком, наприклад, *””*.

*Коментарі*. Це частина тексту програми, що виконує чисто інформаційну функцію, тобто, служить для пояснення використання об'єктів програми і дій над ними. Коментар являє собою певним чином виділену послідовність символів (не обов'язково з алфавіту мови), яка не обробляється компілятором.

Так для мови С/С++ послідовність символів, взята в дужки виду /\* … \*/ використовується для багаторядкових коментарів, а відокремлена символами // - для однорядкових.

Наприклад,

//Це коментар.

Специфічним видом лексем, що використовуються у програмах, зокрема, на мові С/С++, є *управляючі послідовності* (*Esc*-послідовності) - спеціальні символьні комбінації, які використовуються у функціях введення і виведення інформації. Вони, зазвичай, будуються на основі використання символу «\» (обов'язковий перший символ) і комбінації латинських літер і цифр. Наприклад, у мові С/С++ \n - символ нового рядка.

Дві сусідні лексеми повинні бути відокремлені одна від одної одним і декількома роздільниками до числа яких належать пробіл, символи горизонтальної та вертикальної табуляції, символ нового рядка тощо. Виняток - рядки, які можуть включати символ пропуску.

**Елементи даних**.

Елементи даних – це найменші неподільні одиниці даних. Елементами даних, що обробляються в програмах, є *константи* і *змінні*.

***Константи*** - дані, значення яких в процесі роботи програми залишаються незмінними (постійними). Константи характеризуються своїм значенням і, можливо, ім'ям (типізовані константи).

У найпростішому випадку константа являє собою число (123, 2.87), символ (‘A’), літерал ("це рядок"), логічне значення (true, false).

Числа, що використовуються в програмах, можуть бути цілими або дійсними; додатними або від’ємними. Дійсні числа можуть бути представлені у формі з фіксованою або плаваючою крапкою. Більш детально форми запису чисел розглянемо в наступних лекціях при вивченні мови С++.

Цілі числа можуть бути представлені у різних системах числення. Зокрема, у мові Рascal вони можуть задаватися у десятковій або шістнадцятковій системах числення (для позначення шістнадцяткових чисел як префікс використовується символ $, наприклад, $3E816 = 100010). У мові С/С++ цілі числа можуть бути представлені не тільки у десятковій та шістнадцятковій (починаються з префікса 0х або 0Х) системах числення, а й у вісімковій (починаються з префікса нуль - 0). Наприклад, 074, 0x2B.

Константи являють собою фіксовані значення, що не можуть змінюватися впродовж виконання всієї програми.

Спосіб визначення кожної константи залежить від її типу. слід поділяти на літеральні та типізовані.

**Літеральна константа** — це лексема, що являє собою зображення фіксованого числового, рядкового або символьного значення. Такі константи бувають **цілі, дійсні, символьні та рядкові**

|  | Літеральні константи мови |  |
| --- | --- | --- |
| Константа | Формат | Приклади |
| Цiла | Десятковий: послідовність десяткових цифр  (0, 1, 2, 3, 4, 5, 6, 7, 8, 9), що починається не з нуля, якщо це не число нуль | 9, 0, 217925 |
|  | Вісімковий: нуль, за яким розташовані вісімкові цифри (0, 1, 2, 3, 4, 5, 6, 7) | 02, 050, 07245 |
|  | Шістнадцятковий: 0х чи 0Х, за яким розташовані шістнадцяткові цифри (0, 2, 3, 4, 5, 6, 7, 8, 9, А, В, С, D, Е, F) | 0x1B9, 0X00FF |
| Дійсна | Десятковий:[цифри].[цифри]  Експоненціальний: [цифри][.][цифри]{Е|е}[+|-][цифри] | 9.7, .001, 87. 0.7Е6, .15е-3 9.2, 920 е-2, 92.Е-1, .92Е1 |
| Символьна | Один чи два символи, що подаються в апострофах | ‘А’ , ‘ю’ , ‘\* ‘ , ‘db’ , ’\0′, ‘\n’, ’\012′, ‘\х07\х07’ |
| Рядкова | Послідовність символів, що подають­ся в лапках | “RESULT”, |

**Цілі константи** можуть бути десятковими, вісімковими та шiстнадцятковими.

***Змінні*** - це дані, значення яких в процесі роботи програми можуть змінюватися. Змінні характеризуються ім'ям (ідентифікатором) і значенням (початковим, поточним).

***Змінні в*  С++**

**Змінна** — це іменована область пам’яті, у якій зберігаються дані визначеного типу. Змінна має ім’я, розмір та інші атрибути, такі як видимість, час існування тощо. *Ім’я* *змінної* служить для звертання до області пам’яті, у якій зберігається її значення. Кожна змінна повинна мати своє ім’я, причому в одному блоці не може бути двох змінних з однаковим ім’ям. ***Перед використанням будь-яка змінна повинна бути описана***, при цьому для неї резервується деяка область пам’яті, розмір якої залежить від конкретного типу змінної. Під час виконання програми змінна може приймати різні значення. Загальний вигляд опису змінних:

**[клас пам’яті] [const] тип ім’я [ініціювання];**

де необов’язковий *клас пам’яті* може приймати одне зі значень — **auto, extern, static** чи **register** (тут і далі при описі синтаксису об’єктів програмування необов’язкові частини синтаксичних конструкцій мови подано у квадратних дужках «[]»);

модифікатор **const** вказує, що змінна не може змінювати своє значення, у цьому випадку її називають **типізованою (іменованою) константою** або просто **константою**;

**ініціювання** — це присвоювання змінній при описі початкового значення, яке записується зі знаком рівності — **= значення** або в круглих дужках — (**значення**). Константа ***повинна бути ініційована при описі***. Один оператор може містити опис декількох змінних одного типу, розділяючи їх комами, наприклад:

**const int n = 20, m = 5, k = 4;** — ініціювання констант **n, m, k** цілого типу;

**float h = 17.5, d(5.5), sum**; — опис дійсних змінних **h, d, sum**, ініціювання **h і d**;

**char sf = 'f', st[ ] = "Knowledge is power."**; — ініціювання символьних змінних.

Якщо **тип** значення, що ініціюється, не збігається з типом змінної, то виконуються перетворення типу.

**Областю дії ідентифікатора змінної** є частина програми, в якій його можна використовувати для доступу до зв’язаної з ним області пам’яті. Залежно від області дії змінна може бути локальною або глобальною.

***Локальна змінна*** визначена всередині блока (блок розташований між фігурними дужками). Область її дії обмежена початком опису змінної та кінцем блока, включаючи усі вкладені блоки. Змінна, визначена поза будь-яким блоком, називається глобальною, і областю її дії вважається файл, у якому вона визначена від початку опису до його кінця.

**Клас пам’яті** визначає час існування та область видимості програмного об’єкта, тобто змінної. Якщо клас пам’яті не зазначений явно, то він визначається компілятором, виходячи, з контексту оголошення.

***Час існування*** змінної може бути постійним (протягом виконання програми) і тимчасовим (протягом виконання блока).

***Областю видимості*** ідентифікатора називається частина тексту програми, з якої можна здійснити звичайний доступ до зв’язаної з ідентифікатором області пам’яті. Найчастіше область видимості збігається з областю дії. Винятком є ситуація, коли у вкладеному блоці описана змінна з таким же ім’ям. У цьому випадку зовнішня змінна у вкладеному блоці невидима, хоча він і входить до її області дії. Проте до цієї змінної, якщо вона глобальна, можна звернутися, застосовуючи операцію доступу до області видимості — **“::“**.

Клас пам’яті задають такі специфікатори:

* **auto** — автоматична змінна, для якої пам’ять виділяється у стеку і за необхідності ініціюється кожного разу при виконанні оператора, що містить її визначення. Звільнення пам’яті відбувається при виході з блока, де описана змінна. Час її існування — з моменту опису до кінця виконання блока. Для глобальних змінних цей специфікатор не використовується, а для локальних він приймається за замовчуванням, тому можна не задавати явно;
* **extern** означає, що змінна визначена в іншому місці програми (в іншому файлі або далі по тексту) і використовується для створення змінних, доступних в усіх модулях програми, де вони оголошені. При ініціюванні змінної у тому ж операторі, спеціфікатор **extern** ігнорується;
* **static** — статична змінна, що має постійний час існування. Вона ініціюється один раз при першому виконанні оператора, що містить визначення змінної. Залежно від розташування оператора, описані статичні змінні можуть бути глобальними і локальними. Глобальні статичні змінні видимі тільки у тому модулі, в якому вони описані;
* **register** — аналогічний до специфікатора **auto**, але пам’ять виділяється по можливості в регістрах процесора і за відсутності такої можливості у компілятора змінні обробляються як **auto**.

Приклад фрагменту програми з використанням розглянутих вище понять:

**int d;** //1 — глобальна змінна **d**

**int main()**

**{**

**int b;** //2 — локальна змінна **b**

**extern int y;** //3 — змінна **у** визначена в іншому місці програми

**static int s;** //4 — локальна статична змінна **s**

**d = 1;** //5 — присвоювання значення глобальній змінній

**int d;**   //6 — локальна змінна **d**

**d = 10;** //7 — присвоювання значення локальній змінній

**::d = 3;**  //8 — присвоювання значення глобальній змінній

**return 0;**

**}**

**int у = 4;**       // 9 — визначення та ініціювання змінної **у**

У цьому прикладі глобальна змінна **d** визначена поза всіма блоками. Пам’ять для неї виділяється в сегменті даних на початку роботи програми, областю дії є вся програма. Область видимості — вся програма, крім рядків 6-8, тому що в першому з них визначається локальна змінна з тим же ім’ям, область дії якої починається з початку її опису і закінчується при виході з блока. Змінні **b i s** — локальні, область їх видимості — блок, але час існування різний: пам’ять під **b** виділяється в стеку при вході у блок і звільняється при виході з нього, а змінна **s** розташована у сегменті даних та існує увесь час роботи програми. Якщо початкове значення змінних явно не задається, компілятор присвоює глобальним і статичним змінним нульове значення відповідного типу. Автоматичні змінні не ініціюються. ***Початкове ініціювання змінних не є обов’язковим, проте все ж його бажано здійснювати***.

Опис змінної може виконуватися у формі оголошення або визначення. Оголошення інформує компілятор про тип змінної і класи пам’яті, а **визначення** містить, крім цього, вказівку компілятору про виділення пам’яті відповідно до типу змінної. У C++ більшість оголошень є одночасно і визначеннями (у наведеному вище програмному фрагменті тільки опис **extern int у**; є оголошенням, але не визначенням). ***Змінна може бути оголошена багаторазово, а визначена тільки в одному місці програми***, оскільки оголошення тільки описує властивості змінної, а визначення зв’язує її з конкретною областю пам’яті.

**Операції**.

Для задання дій над даними служать *операції*. Вони поділяються на *унарні* і *бінарні*. У*нарні* операції мають тільки один операнд (елемент даних), перед яким вказується символ операції (*-*7*, not false*). Більшість же операцій є *бінарними*, тобто таких, які містять два операнда (8 *+ с, k* /9).

За характером виконуваних дій операції можна розділити на декілька груп:

- арифметичні,

- порівняння,

- логічні,

- символьні тощо.

*Арифметичні операції*. Застосовуються до числових операндів. Їх перелік для мови С/С++ наведений у табл.1.

Таблиця 1.Арифметичні операції мови С/С++

|  |  |  |  |
| --- | --- | --- | --- |
| Знак операції | Найменування операції | Приклад | Особливості реалізації |
| *+* | додавання |  |  |
| - | віднімання |  |  |
| \* | множення |  |  |
| / | цілочисельне ділення | 5 / 2 = 2 5. / 2 = 2.5 | Результатом ділення цілих чисел буде цілочисельне значення, що дорівнює цілій частині результату ділення першого операнда на другий (дробова частина відкидається); результат ділення дійсних чисел – дійсне число |
| % | ділення з залишком (залишок від ділення) | 11 *%* 5 *=* 1 | Результатом операції *%* є цілочисельне значення, що дорівнює залишку, отриманому при цілочисельному діленні |

*Операції порівняння*. Застосовуються до числових або символьних операндів. Результатом цих операцій є логічне значення "істина" або "хибність". Перелік операцій порівняння для мови С/С++ наведений у табл.2.

Таблиця 2.Операції порівняння мови С/С++

|  |  |
| --- | --- |
| Знак операції | Найменування операції |
| == | дорівнює |
| != | не дорівнює |
| *<* | менше |
| *>* | більше |
| *<=* | не більше |
| *>=* | не менше |

*Логічні операції*. Застосовуються до логічних значень *істина* (*true*) або *хибність* (*false*). Їх перелік для мови С/С++ наведений у табл.3.

Таблиця 3.Логічні операції мови C/С++

|  |  |  |  |
| --- | --- | --- | --- |
| Знак операції | Найменування операції | Приклад | Особливості реалізації |
| ! | НЕ (заперечення) | !a | Результат - протилежний значенню операнда |
| && | І (логічне множення) | a&&b | Результат істина, якщо всі операнди істині |
| || | АБО (логічне додавання) | a||b | Результат істина, якщо хоча б один з операторів - істина |

Дані логічні дії виконуються за правилами булевої алгебри.

Основне призначення логічних операцій - побудова складних умов:

! - змінює зміст логічного операнда на протилежний;

&& - використовується для перевірки істиності декількох компонент;

|| - служить для перевірки істиності хоча б однієї із компонент.

Символьні операції. Застосовуються до літералів або до символів. Визначені не у кожній мови програмування. Так у мові C/С++ такі операції не передбачені, у мові Рascal є одна така операція (табл. 4).

Таблиця 4.Символьні операції Рascal

|  |  |  |
| --- | --- | --- |
| Знак операції | Найменування операції | Приклад |
| *+* | Конкатенація | ‘A’ + ‘CD’ = ‘ACD’ |

*Бітові (порозрядні) операції*. Операції роботи з бітами. Характерні не для кожної мови програмування. Застосовуються лише до цілочисельних операндів. Їх перелік для мов С/С++, С#, Ruby, Java представлений у табл. 5.

Таблиця 5. Бітові операції мов С/С++, С#, Ruby, Java

|  |  |  |  |
| --- | --- | --- | --- |
| Знак операції | Найменування операції | Приклад | Особливості реалізації |
| ~ | Доповнення (заперечення бітів) | ~6=-7 | ~6= ~0 0000110= 1 1111001(д) = 1 1111000(о) = 1 0000101(п) = -7 |
| & | І (логічне множення) | 3&2=5 | 1&1=1, 0&1=0, 1&0=0, 0&0=0 |
| | | АБО (логічне додавання) | 3|2=3 | 1|1=1, 0|1=1, 1|0=0, 0|0=0 |
| ^ | виключаюче АБО | 3^2=1 | 1^1=0, 0^1=1, 1^0=1, 0^0=0 |
| << | Зсув вліво на задане число бітів | a<<b | Зсув бітів числа а ліворуч на b позицій (здвиг вліво на 1 розряд збільшує число в 2 рази, оскільки кожен i-й розряд має вагу 2і):00000111<<1 = 00001110 |
| >> | Зсув вправо на задане число бітів | a>>b | Зсув бітів числа а праворуч на b позицій (здвиг вправо на 1 розряд зменшує число в 2 рази): 00000111>>1 = 00000011 |

При реалізації операцій зсуву біти, що звільняються, заповнюються нулями, а значення бітів, що вийшли за межі початкової величини, втрачаються.

*Операції приросту та спаду*. Мова C/С++ має також оператори приросту та спаду змінних. Оператор приросту ++ додає 1 до свого операнду, тоді як --, навпаки, віднімає 1.

Операції приросту / інкременту (++) і спаду / декременту (--) можуть використовуватись як префіксні оператори (перед змінною, наприклад, ++n), так і як постфіксні (після змінної, наприклад, n++). В обох випадках, як наслідок, збільшується значення n. Але вираз ++n збільшує n до того, як це значення буде використане, тоді як n++ збільшує n після того, як було використане початкове значення. Це означає, що в контексті, де дійсно використовується значення, а не тільки самий ефект, ++n і n++ — відмінні. Так, якщо n дорівнює 5, тоді

x = n++;

присвоїть x значення 5, зате у випадку

x = ++n;

x дорівнюватиме вже 6. В обох випадках, n стане рівним 6.

Оператори приросту та спаду можуть використовуватись тільки зі змінними; вирази на кшталт (i+j)++ заборонені.

**Вирази**.

***Ви́раз*** — це синтаксична конструкція мови, яка складається із даних (операндів) та знаків операцій і служить для обчислення значення невідомої величини.

У залежності від типів операндів і операцій розрізняють *арифметичні*, *логічні*, *символьні* вирази, вирази *відношення* та інші. Наприклад, (z > a1) and (z < b1) - умова перевірки приналежність змінної *z* інтервалу (a1, b1).

Послідовність виконання операцій у виразі визначається трьома факторами:

- пріоритетом операцій;

- порядком розташування операцій у виразі;

- використанням дужок.

Пріоритет операцій у різних мовах програмування дещо різний. Зокрема, у С/С++ пріоритет операцій (табл. 6) відношення більший, аніж логічних операцій, а у Рascal - навпаки.

Таблиця 6.Пріоритет операцій у мові С

|  |  |  |
| --- | --- | --- |
| Пріоритет | Операції | Категорія операцій |
| Перший (вищий) | ~ ++ -- + - | унарні операції |
|  | \* / % | бінарні операції |
|  | + - | бінарні операції |
|  | << >> | бінарні операції |
|  | < <= > >= | бінарні операції |
|  | == != | бінарні операції |
|  | & | бінарні операції |
|  | ^ | бінарні операції |
|  | | | бінарні операції |
|  | && | бінарні операції |
|  | || | бінарні операції |

**Обробка виразів**

Обробка виразів здійснюється у відповідності з наступними правилами:

1. Операції з більш високим пріоритетом виконуються в першу чергу.

2. Якщо операції у вираз мають однаковий пріоритет, то вони виконуються послідовно зліва направо.

3. Частина виразу в круглих дужках виконується в першу чергу.

Окрім констант і змінних до числа операндів виразу відносяться *функції* - деяка обчислювальна процедура, на вхід якої передається набір допустимих аргументів, а на виході отримується єдиний результат - значення якого-небудь типу.

Подібно змінним, кожна функція має своє унікальне ім'я (ідентифікатор). Аргументами функції можуть бути будь-які вирази, але типи аргументів і їхня кількість визначаються ім'ям функції.

Кожна функція реалізується за допомогою підпрограми, що є або стандартною, або створеною користувачем. Відповідно до цього всі функції поділяються на *стандартні* та *функції користувача*.

Основні стандартні математичні функції мови С/С++:

|  |  |
| --- | --- |
| *Функція* | *Опис* |
| abs (x) | ціле абсолютне значення аргумента *x*, що може бути дійсним числом (формується шляхом відкидання дробової частини *x*) |
| fabs (x) | дійсне абсолютне значення аргумента *x*, що може бути дійсним числом |
| sin(x) | синус числа *x* |
| cos(x) | косинус числа *x* |
| tan(x) | тангенс числа *x* |
| exp(x) | експонента числа *x* (ех) |
| log (x) | натуральний логарифм числа *x* |
| sqrt(x) | квадратний корінь з числа x |
| ceil(x) | округлення аргумента x до найближчого більшого цілого |
| floor (x) | округлення аргумента х до найближчого меншого цілого (шляхом відкидання його дробової частини) |
| pow(x,y) | піднесення аргумента х до ступеню y |

**Оператори** (інструкції). ***Оператор*** - це команда мови програмування, якою задається певний крок процесу обробки інформації на ЕОМ. Оператор вказує на те, яку операцію потрібно здійснити. Також часто в українській технічній літературі операторами називають окремі види [*інструкцій*](https://uk.wikipedia.org/wiki/%D0%86%D0%BD%D1%81%D1%82%D1%80%D1%83%D0%BA%D1%86%D1%96%D1%8F_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)), такі як цикли й умовні інструкції. Ця плутанина виникла через хибний переклад частиною перекладачів англ. *statement* — «інструкція» як «оператор». Цьому також сприяли відмінності в термінології різних мов програмування.

***Для самостійного вивчення*:** Поглибити матеріал лекції за наданою літературою. Вивчення лекційного матеріалу та додаткових джерел. Розгляд запитань і виконання завдань для самостійної роботи, запропонованих на лекції.

***Контрольні запитання*.**

1. З яких елементарних конструкцій складається мова програмування?
2. Які основні складові визначають мову програмування?
3. Що визначає семантика?
4. З яких елементів складається опис мови?
5. З якою метою використовуються нотації Бекуса-Наура та синтаксичні діаграми?
6. Що таке лексема?
7. Для чого в мовах програмування використовуються зарезервовані (ключові) слова?
8. Що таке літерал?
9. Що таке управляючі послідовності?
10. Які ви можете визначити типи операцій?
11. Які ви можете визначити типи виразів? Від чого залежить тип виразу?
12. Що таке оператор?

***Література***

1. Ковалюк Т. В. Алгоритмізація та програмування: Підручник. — Львів: «Магнолія 2006», 2013. — 400 с., ил.
2. Джейс Либерти Освой самостоятельно С++ за 21 день: 3-е изд. пер. с англ.: Уч. пос. – М.: Издательский дом „Вильямс”, 2001. – 816 с.: ил..
3. Куприянова Л.М. Программирование, алгоритмические языки и вычислительная математика: Учеб. пособие. — М.: Финансы и статистика, 1985. —223 с.