WPROWADZENIE
DO PROGRAMOWANIA

Podstawy Kodu

CZĘŚĆ 2

Na początek - system dziesiętny

Dla nas, ludzi naturalnym sposobem prezentacji liczb jest system dziesiętny. Oznacza to, że wyróżniamy dziesięć cytr. Są nimi: zero, jeden, dwa, trzy, cztery, pięć, sześć, siedem, osiem oraz dziewięć. Oznacza się je odpowiednio: 0, 1, 2, 3, 4, 5, 6, 7, 8 oraz 9. Jak widać, wliczając zero, jest ich dziesięć. Spróbujcie uświadomić sobie, że liczenie jest tylko i wyłącznie ILOŚCIĄ, a nie zapisem liczb. Zapis dziesiętny powstał wieki temu, prawdopodobnie, dlatego, że mamy dziesięć palców. Jednakże, nie będziemy teraz się zajmować historią.

Przejdźmy zatem do bardziej konkretnych rzeczy. Umiemy już policzyć do dziecięciu, wliczając liczbę zero. Natomiast co się stanie, gdy będziemy mieli do policzenia jakąś większą ilość? Otóż, przeskakujemy automatycznie, na następną pozycję, a cyfry zwiększmy tylko na pozycji wysuniętej najbardziej w prawo. Właśnie ta najbardziej w prawo wysunięta pozycja jest najsłabsza, a najbardziej w lewo - najmocniejsza. Tym sposobem znowu zwiększamy cyfry, aż uzyskamy dziewięć. Następna liczba, przesunie cyfrę, która znajduje się o jedną pozycję w lewo. Natomiast gdy już nawet dziewiątka będzie na najbardziej w lewo wysuniętej pozycji, dodajemy nową pozycję. Cykl zaczyna się ponownie i tak w nieskończoność.

Weźmy na przykład liczbę 274, czyli dwieście siedemdziesiąt cztery. Na najsłabszej pozycji widnieje cyfra 4. Pozycja ta nosi nazwę pozycji jedności. Mamy zatem 4 jedności. Na drugiej pozycji jest cyfra 7. Cyfra ta znajduje się na drugiej pozycji, czyli pozycji dziesiątek. Można więc powiedzieć, że jest tam siedem dziesiątek, inaczej mówiąc 70 jedności. Na trzeciej natomiast pozycji jest cyfra 2. Trzecia pozycja to pozycja setek, czyli mam dwie setki. Innymi słowy, liczba 274 to dwie setki, siedem dziesiątek i 4 jedności. Można to zapisać następująco: 4*1 + 7*10 + 2*100. Po dokonaniu tegoż działania, wyjdzie 274. Czas, aby się temu działaniu przyjrzeć. Jak widać, każdy kolejny składnik zawiera cyfrę z powyższej liczby oraz ciągle zwiększający mnożnik. Mnożnik ten najpierw jest równy 1, potem 10, a na końcu 100. Znaczy to, że każdy następny jest pomnożony przez 10. Można więc zapisać to jeszcze inaczej. Liczba 274 to tak jak: $4*10^0 + 7*10^1 + 2*10^2$. Jak widzimy, mnożnik to liczba 10 z ciągle zwiększającą się potęgą.

Ta informacja przyda się w następnych działach omawiających przeliczanie z jednego systemu na drugi. Zwróćmy uwagę teraz na rzecz, która chociaż trochę uzmysłowi wam, jak działa system dziesiętny. Gdybyśmy chcieli zwiększyć o 1 liczbę 347, to zawsze, automatycznie zwiększmy cyfrę, która znajduję się na pozycji wysuniętej najdalej w prawo. Powstanie zatem 348. Natomiast, gdy chcemy zwiększyć o 1 liczbę 429, widzimy, że nie można już nic do 9-tki dodać, gdyż nie ma już wyższej cyfry. Co wtedy robimy? - każdy wie. Zwiększamy o jeden wartość cyfry znajdującej się na pozycji z lewej strony, natomiast wartość jedności zerujemy (dajemy najniższą możliwą wartość). Powstaje zatem 430. Jeżeli natomiast chcielibyśmy zwiększyć wartość o 1 liczby 999, to widać, że : nie można zwiększyć jedności, nie można zwiększyć dziesiątek i nie można zwiększyć setek. Dodajemy zatem następną pozycję. Powstanie więc 1000.

System ósemkowy

Skoro powstał system dziesiętny, można wymyślać dowolne systemy liczenia (na przykład czwórkowy itd.). Właśnie jednym z takich systemów jest system ósemkowy. Początkowo był on trochę stosowany, obecnie jednak jego zastosowanie jest znikome. Posłuży nam on jako dobry przykład. Jak się pewnie domyślacie, w systemie tym jest osiem cyfr. Wcale się nie mylicie. Są to: 0, 1, 2, 3, 4, 5, 6 oraz 7. Jest ich, więc 8, stąd nazwa. Działa on na tej samej zasadzie, co system dziesiętny. To znaczy, że gdy już jakaś cyfra jest na maksymalnej wartości, zwiększamy cyfrę na następnej pozycji. Wyjaśni to się na przykładzie. Przekształcajmy kolejne liczby i zobaczmy, jakie są różnice. Liczba zero (0) tak samo wygląda w obu systemach. Tak samo ma się sytuacja z jedynką (1), dwójką (2), trójką (3) itd. Sytuacja staje się skomplikowana, gdy dojedziemy do siódemki (7). Liczba 7 wygląda tak samo w obu systemach. Jednak nadchodzi następna liczba, zwana przez nas jako osiem. System ósemkowy nie zna takiej cyfry, więc powstaje następna pozycja. Zatem liczba osiem (8) w systemie dziesiętnym to liczba dziesięć (10) w systemie ósemkowym. Była to bardzo ważna konwersja i dobrze by było, gdybyś ją zrozumiał. Liczby takie jak: 6, 7, 8, 9, 10, w systemie ósemkowym będą wyglądać odpowiednio: 6, 7, 10, 11, 12.

Gdybyśmy chcieli sprawdzić, czy rzeczywiście liczba na przykład 14 w systemie ósemkowym to 12 w dziesiętnym, musimy przeprowadzić konwersję. Dokonuje tego tak, jak robiliśmy to w akapicie o systemie dziesiętnym, z tym, że podstawą mnożenia będzie liczba 8. Zatem, rozpisujemy liczbę 14 (s. ósemkowy) w następujący sposób. Jest to 4*1 + 1*8, czyli 4+8 czyli 12. Innymi słowy, jest to $4*8^0 + 1*8^1$. Po policzeniu wyniku muszą się zgadzać.

Zauważcie, że w systemie dziesiętnym kolejne pozycje miały wartości: 1, 10, 100, 1000, 100000, 1000000 itd., ponieważ podstawą była liczba 10. W systemie ósemkowym podstawą jest liczba 8, a kolejne pozycje wyglądają następująco: 1, 8, 64, 512, 4096, 32768 itd.

System dwójkowy

Powiedzieliśmy sobie, że można wymyślać dowolny system zapisu liczb. Skoro tak, to, czemu miałby nie powstać system dwójkowy, składający się tylko z dwóch cyfr: 0 (zero) i 1 (jeden). Działa on analogicznie tak samo jak poprzednie systemy. Wyjaśni się zaraz wszystko na konkretnym przykładzie. Weźmy na przykład kilka pierwszych liczb naszego systemu dziesiętnego. Będziemy je konwertować na system dwójkowy, zwany również binarnym. Pierwsza liczba w naszym systemie to 0 (zero). W systemie dwójkowym, liczba ta również jest równa 0, gdyż istnieje tam taka cyfra. Kolejna liczba to 1 (jeden). W systemie dwójkowym, również taka cyfra istnieje, więc zapisujemy 1. Kolejna liczba to 2 (dwa). Wiemy, że nie istnieje tam taka cyfra, więc dodajemy kolejną pozycję, a pozycję wysuniętą na prawo, zerujemy. Zatem liczba 2 w systemie dziesiętnym ma postać "10" w systemie dwójkowym. Bynajmniej nie jest to "dziesięć" tylko "jeden, zero". Kolejne liczby w systemie dziesiętnym to: 3, 4, 5, 6, 7, 8, 9 itd. W systemie dwójkowym wyglądają one odpowiednio: 11, 100, 101, 110, 111, 1000, 1001. Jak widzimy, zasada jest cały czas taka sama.

Zanim zaczniemy uczyć się, jak w prosty sposób zamienić liczbę z jednego systemu na drugi, postawmy sobie pytanie: Po co komputerowi taki system?

No więc, jak zapewne wszyscy wiedzą, komputer składa się z części elektronicznych. Wymiana informacji polega na odpowiednim przesyłaniem sygnałów. Podstawą elektroniki jest prąd elektryczny, który w układach elektronicznych albo płynie albo nie. Zatem, aby łatwiej było komputerowi rozpoznawać sygnały, interpretuje on płynący prąd jako "1" (jeden), a jego brak jako "0" (zero). Nie trudno się domyślić, że komputer operując odpowiednim ustawieniem, kiedy ma płynąc prąd, a kiedy nie ustawia różne wartości zer i jedynek. Procesor konwertuje je na liczby i w ten sposób powstają czytelne dla nas obrazy, teksty, dźwięk itd. Nie tylko w postaci sygnałów elektrycznych reprezentowane mogą być zera lub jedynki. Również na wszelkich nośnikach, np. płyta CD, na której nagrywarka wypala malutkie wgłębienia. Właśnie te wgłębienia są jedynkami, a "równiny" zerami (albo i odwrotnie). Zatem podsumujmy: komputer zna tylko zera i jedynki. Bity przyjmują tylko jedną z tych dwóch wartości. Osiem bitów to jeden bajt. Ustawienie ośmiu bitów decyduje o numerze, który może przyjąć maksymalnie 256. Numer decyduje o znaku, jaki komputer ma wykorzystać.

Konwersja liczby dwójkowej (binarnej) na dziesiętną

Skoro już wiesz, po co nam system binarny, dowiesz się jak przeliczać go na nasz system dziesiętny. Otóż nie jest to zbyt skomplikowane. Przypomnijcie sobie sposób z liczbami w systemie ósemkowym. Tu oczywiście robimy to analogicznie tak samo, z tym, że podstawą jest naturalnie liczba 2. Weźmy sobie zatem jakąś liczbę zapisaną w systemie dwójkowym, np. 1000011. Jak już wcześniej mówiliśmy, zaczynamy od cyfr najsłabszych, czyli wysuniętych najbardziej na prawo. Najbardziej na prawo wysunięta jest cyfra 1, a więc tak jak poprzednio mnożymy ją przez podstawę systemu z odpowiednią potęgą. Podstawą systemu jest 2. Zatem, cała konwersja ma postać: $1*2^0 + 1*2^1 + 0*2^2 + 0*2^3 + 0*2^4 + 0*2^5 + 1*2^6$, a to się równa: 1 + 2 + 0 + 0 + 0 + 0 + 64, czyli jest to 67 w systemie dziesiętnym. Moje gratulację - udało się skonwertować liczbę w zapisie dwójkowym na zapis dziesiętny. Teraz dobrze by było gdybyś przeanalizował sobie dokładnie powyższą konwersję.

Konwersja liczby dziesiętnej na dwójkową (binarną)

Teraz, skoro już umiesz konwertować liczby z zapisu dwójkowego na dziesiętny warto by było skonwertować je odwrotnie, to znaczy z zapisu dziesiętnego na dwójkowy. Gdybyśmy liczyli na piechotę, byśmy musieli sprawdzać kolejne wielokrotności liczby 2. Sposób ten raczej jest mało stosowany, zajmijmy się trochę lepszym. Jest to prosty sposób, wcale nie wymaga myślenia. Najpierw bierzemy liczbę, jaką chcemy skonwertować na zapis dwójkowy. Weźmy liczbę z poprzedniego rozdziału i sprawdźmy, czy nam się to zgadza. Zatem, liczba którą będziemy konwertować to 67. Sposób jest następujący: liczbę dzielimy przez 2 i jeżeli wynik będzie z resztą: zapisujemy 1, jeżeli nie - zapisujemy 0. Następnie znowu dzielimy przez 2 to co zostało z liczby, ale bez reszty. Taki proces trwa, aż zostanie 0 (zero). Otrzymane zera i jedynki zapisujemy w odwrotnej kolejności. Wyjaśni się to wszystko na konkretnym przykładzie. Zatem do dzieła:

67	:2	1
33	:2	1
16	:2	0
8	:2	0
4	:2	0
2	:2	0
1	:2	1

Co daje 1000011. Jak widzimy, wynik zgadza się. Widać również, że zawsze na samym końcu po podzieleniu będzie 0, zatem ostatnia liczba jest równa 1. Jeden podzielić na dwa zawsze wyjdzie 0,5 zatem wynik z resztą. Co za tym idzie - pierwsza cyfra w zapisie dwójkowym jest ZAWSZE RÓWNA 1. Nie tylko matematycznie można to udowodnić. W elektronice, również musi być taka postać rzeczy. Przyjęliśmy bowiem, że dla komputera brak przepływu prądu oznacza "0", natomiast przepływ prądu - "1". Sygnał zatem nie może zaczynać się od "0", gdyż jest to brak sygnału. Procesor nie wie, czy sygnał już się zaczął, czy jeszcze nie. Początek musi być "1" (jest sygnał).

Zmienne

Konstrukcja programistyczna posiada trzy podstawowe atrybuty:
- symboliczną nazwę
- miejsce przechowywania
- wartość

Zmienna pozwala w kodzie źródłowym odwoływać się przy pomocy nazwy do wartości lub miejsca jej przechowywania. Nazwa służy do identyfikowania zmiennej w związku z tym często nazywana jest identyfikatorem. Miejsce przechowywania przeważnie znajduje się w pamięci komputera i określane jest przez adres i długość danych. Wartość to zawartość miejsca przechowywania. Zmienna zazwyczaj posiada również czwarty atrybut: typ, określający rodzaj danych przechowywanych w zmiennej i co za tym idzie sposób reprezentacji wartości w miejscu przechowywania. W programie wartość zmiennej może być odczytywana lub zastępowana nową wartością, tak więc wartość zmiennej może zmieniać się w trakcie wykonywania programu, natomiast dwa pierwsze atrybuty (nazwa i miejsce przechowywania) nie zmieniają się w trakcie istnienia zmiennej. W zależności od rodzaju języka typ może być stały lub zmienny. Konstrukcją podobną lecz nie pozwalającą na modyfikowanie wartości jest stała.

Inaczej wygląda zmienna w programowaniu funkcyjnym (gdzie idea zmiennej jest zbliżona do zmiennej matematycznej). Podczas wchodzenia obliczeń do kontekstu, w którym zmienna jest związana, jest jej nadawana wartość, która nie zmienia się, aż do opuszczenia kontekstu. Jednak przy ponownym wejściu w ten kontekst, zmiennej może być przypisana inna wartość niż poprzednio.

Typ zmiennej

W językach ze statycznym typowaniem zmienna ma określony typ danych jakie może przechowywać. Jest on wykorzystywany do określenia reprezentacji wartości w pamięci, kontrolowania poprawności operacji wykonywanych na zmiennej (kontrola typów) oraz konwersji danych jednego typu na inny.

W językach z typowaniem dynamicznym typ nie jest atrybutem zmiennej lecz wartości w niej przechowywanej. Zmienna może wtedy w różnych momentach pracy programu przechowywać dane innego typu.

Deklaracja i definicja

Deklaracja zmiennej to stwierdzenie, że dany identyfikator jest zmienną, przeważnie też określa typ zmiennej. W zależności od języka programowania deklaracja może być obligatoryjna, opcjonalna lub nie występować wcale. Definicja oprócz tego, że deklaruje zmienną to przydziela jej pamięć. Podczas definiowania lub deklarowania zmiennej można określić jej dodatkowe atrybuty wpływające na sposób i miejsce alokacji, czas życia, zasięg i inne.

Zasięg, czas życia, widoczność

Zasięg zmiennej określa, gdzie w treści programu zmienna może być wykorzystana, natomiast czas życia zmiennej to okresy w trakcie wykonywania programu, gdy zmienna ma przydzieloną pamięć i posiada (niekoniecznie określoną) wartość. Precyzyjnie zasięg odnosi się do nazwy zmiennej i przeważnie jest aspektem leksykalnym, natomiast czas życia do zmiennej samej w sobie i związany jest z wykonywaniem programu. Ze względu na zasięg można wyróżnić podstawowe typy zmiennych:

- globalne - obejmujące zasięgiem cały program,
- lokalne - o zasięgu obejmującym pewien blok, podprogram. W językach obsługujących rekurencję zazwyczaj są to zmienne automatyczne, natomiast w językach bez rekurencji mogą być statyczne.

Zmienne zadeklarowane w module mogą być zmiennymi prywatnymi modułu - dostępne wyłącznie z jego wnętrza lub zmiennymi publicznymi (eksportowanymi) dostępnymi tam gdzie moduł jest wykorzystywany. Podobnie ze zmiennymi w klasie mogą być dostępne:

- tylko dla danej klasy (zmienna prywatna),
- dla danej klasy i jej potomków (zmienna chroniona),
- w całym programie (zmienna publiczna),
- inne ograniczenia w zależności od języka (np. friend czy internal w .net)

Zmienne mogą zmieniać swój pierwotny zasięg np. poprzez importowanie/włącznie do zasięgu globalnego modułów, pakietów czy przestrzeni nazw.

Ze względu na czas życia i sposób alokacji zmienna może być:

- Statyczna - gdy pamięć dla niej rezerwowana jest w momencie ładowania programu; takimi zmiennymi są zmienne globalne, zmienne statyczne w klasie (współdzielone przez wszystkie obiekty klasy, a nawet dostępne spoza klasy), statyczne zmienne lokalne funkcji (współdzielone pomiędzy poszczególnymi wywołaniami funkcji i zachowujące wartość po zakończeniu).
- Automatyczna, dynamiczna - gdy pamięć przydzielana jest w trakcie działania programu ale automatycznie. Są to przeważnie zmienne lokalne podprogramów i ich parametry formalne. Przeważnie alokowane na stosie w rekordzie aktywacji, znikają po zakończeniu podprogramu.
- Dynamiczna - alokowanie ręcznie w trakcie wykonywania programu przy pomocy specjalnych konstrukcji lub funkcji (malloc, new). W zależności od języka zwalnianie pamięci może być ręczne lub automatyczne. Zazwyczaj nie posiada własnej nazwy, lecz odwoływać się do niej trzeba przy pomocy wskaźnika, referencji lub zmiennej o semantyce referencyjnej.

Integer

Liczby całkowite – typ danych dotyczący liczb całkowitych. Liczby te mogą zostać zapisane w pamięci komputera w rozmaity sposób. Obecnie dla liczb naturalnych najczęściej spotykany jest pozycyjny dwójkowy system liczbowy. Inne znane sposoby zapisu to kod Graya i BCD.

Na pamięć komputera można spojrzeć jak na komórki. W każdej z nich trzymany jest elementarny kwant informacji zwany bitem. W praktyce komórki te grupuje się w większe całości zwane, w zależności od rozmiaru, bajtami (8 bitów), słowami (zawierającymi 2 lub więcej bajtów) lub jeszcze inaczej.

liczba bitów	nazwa	zakres
8	char	−128 — +127 (ze znakiem) 0 — +255 (bez znaku)
16	short int	−32 768 — +32 767 (ze znakiem) 0 — +65 535 (bez znaku)
32	int, long int	−2 147 483 648 — +2 147 483 647 (ze znakiem) 0 — +4 294 967 295 (bez znaku)
64	long long int	−9 223 372 036 854 775 808 — +9 223 372 036 854 775 807 (ze znakiem) 0 — +18 446 744 073 709 551 615 (bez znaku)

Liczba zmiennoprzecinkowa

Reprezentacja liczby rzeczywistej zapisanej za pomocą notacji naukowej. Ze względu na wygodę operowania na takich liczbach, przyjmuje się ograniczony zakres na mantysę i cechę. Powoduje to, że reprezentacja liczby rzeczywistej jest tylko przybliżona, a jedna liczba zmiennoprzecinkowa może reprezentować różne liczby rzeczywiste z pewnego zakresu.

String

Tekstowy typ danych (ang. String) – typ danych służący do przechowywania ciągu znaków (zmiennych łańcuchowych). String można tłumaczyć jako „ciąg znaków", ale także „łańcuch", dlatego używa się również określenia „łańcuch znaków" lub krótko „łańcuch". Etymologicznie odpowiada słowu „strzęp", również dłużej ugruntowanemu frazeologicznie.

W niektórych językach programowania jak np. Pascal czy PHP łańcuchy są typem wbudowanym; w pozostałych jak C, C++, Java realizuje się je za pomocą innych struktur języka.

W tradycyjnych realizacjach Pascala (np. Turbo Pascal) zmienna typu String może przechowywać do 255 znaków, a w definicji String[długość] może przechowywać do długość znaków, a ma rozmiar długość+1 i jest zaimplementowana jako tablica, której element o indeksie 0 przechowuje liczbę znaków w tym ciągu (typu bajt więc maksymalna długość łańcucha wynosi 255). W późniejszych implementacjach tego języka (np. Object Pascal w Delphi) dodano inną formę reprezentacji łańcucha, w którym maksymalna długość wynosi 2^32 bajtów i jest dynamicznie przydzielana w zależności od długości napisu, typ ten jest rozszerzeniem sposobu implementacji używanego przez C, dodano również typ w stylu C (PChar).

W C++ oprócz tradycyjnych ciągów znaków w stylu C istnieje w bibliotece standardowej klasa std::string. Ukrywa ona niewygodne aspekty używania napisów w stylu C: zarządzanie pamięcią, określanie długości, łączenie napisów, wstawianie, usuwanie i inne manipulacje na napisie. Dodatkowo pozbyto się problemu znaku kończącego – znak o kodzie \0 może być elementem napisu std::string (długość przechowywana jest oddzielnie). Ponieważ biblioteka standardowa (bazująca w tym zakresie na STL) została dość późno dołączona do oficjalnego standardu, wiele kompilatorów dostarcza własne implementacje typów napisowych – np. String, AnsiString, CString. Również niektóre starsze biblioteki (jak np. Qt) dostarczają własnych typów obsługi napisów.

Tablice

Kontener uporządkowanych danych takiego samego typu, w którym poszczególne elementy dostępne są za pomocą kluczy (indeksu). Indeks najczęściej przyjmuje wartości numeryczne. Rozmiar tablicy jest albo ustalony z góry (tablice statyczne), albo może się zmieniać w trakcie wykonywania programu (tablice dynamiczne). Tablice jednowymiarowe mogą przechowywać inne tablice, dzięki czemu uzyskuje się tablice wielowymiarowe. W tablicach wielowymiarowych poszczególne elementy są adresowane przez ciąg indeksów.

Praktycznie wszystkie języki programowania obsługują tablice – jedynie w niektórych językach funkcyjnych zamiast tablic używane są listy (choć tablice zwykle też są dostępne). W matematyce odpowiednikiem tablicy jednowymiarowej jest ciąg, a tablicy dwuwymiarowej - macierz.

Boolean

Typ logiczny, typ boolowski (ang. boolean) – uporządkowany zbiór wartości logicznych, składający się z dokładnie dwóch elementów: prawda (true, 1, +) i fałsz (false, 0, -), wraz z towarzyszącymi im zdefiniowanymi operatorami standardowymi. Nazwa pochodzi od angielskiego pioniera logiki, matematyka George'a Boole'a.

Typ logiczny stosuje się jako:
- wynik funkcji (zachodzi, bądź nie zachodzi),
- flagę,
- warunek instrukcji warunkowej,
- warunek wyjścia/kontynuowania pętli,
- warunek wykonania skoku.

Różnice pomiędzy typem logicznym a typem liczbowym

Ze względu na to, iż każdą funkcję logiczną da się zapisać funkcją dającą jako wynik liczbę naturalną (fałsz - 0, prawda - liczba większa od zera), zaś operatory logiczne da się zapisać za pomocą mnożenia i dodawania, w niektórych językach programowania, np. w C (do C99), typ logiczny nie występował. Są jednak przyczyny, dla których typ logiczny jest stosowany:
- brak zakresu (nie ma efektu przekroczenia zakresu poprzez sumowanie dwóch warunków prawdziwych),
- oszczędność pamięci,
- większa czytelność kodu.

Pusty typ danych

Typ danych, którego zmienna niesie zerową informację, czyli, zgodnie z teorią informacji, matematyczna klasa wszystkich wartości zmiennych tego typu zawiera dokładnie jeden element. Wprowadzenie takiego typu do systemu typów języka programowania umożliwia pewne rodzaje uogólnień - nie trzeba rozróżniać funkcji, które zwracają wartość, od tych, które

jej nie zwracają (czyli zwracają wartość typu pustego), oraz funkcji, które pobierają jakiś argument, od tych, które tego nie robią.
Typ pusty stosowany jest głównie do:

- Wskazania, że funkcja nie zwraca wyniku (języki C, C++).
- Wskazania, że funkcja nie pobiera żadnych argumentów (konieczne w języku C, opcjonalne w C++)
- Definiowania wskaźników na dane nieokreślonego typu (języki C i C++).

Operatory

Do podstawowych operatorów, będących elementem większości języków programowania, należą operatory arytmetyczne: dodawania (+), odejmowania (-), mnożenia (*), dzielenia (/); operatory porównania: większe niż (>), mniejsze niż (<), większe równe (>=), mniejsze równe (<=), równe (= lub ==), różne (<> lub !=), a także operatory operacji logicznych, operacji bitowych, przypisań itd. Główne cechy opisujące operator to liczba i typy argumentów, typ wartości zwracanej, wykonywane działanie, priorytet oraz łączność lub jej brak oraz umiejscowienie operatora względem operandów. Dany język posiada swoją listę operatorów wraz z określonymi cechami, mówiącymi o kolejności wykonywania operacji w przypadku, gdy nie zastosowano nawiasów. W niektórych językach można definiować nowe operatory oraz zmieniać priorytety i łączność.

Inkrementacja ++ i dekrementacja - -

Inkrementacja i dekrementacja za pomocą operatorów ++ oraz -- to zwiększenie lub zmniejszenie wartości zmiennej o jeden. Operatory te możesz umieszczać przed i po zmiennej.

Najczęściej używana forma:

- **inkrementacja**:
++a; - preinkrementacja
a++; - postinkrementacja

- **dekrementacja**
--b; - predekrementacja
b--; - postdekrementacja

W tym przypadku umieszczenie operatorów przed, czy po zmiennej nie ma znaczenia. Po prostu w danej linii kodu zmieniasz wartość zmiennej o jeden.

Są jednak przypadki gdzie kolejność ma znaczenie. Zacznijmy od:

++zmienna
--zmienna

Umieszczenie operatora przed zmienną skutkuje pierwszeństwem wykonania operatora względem pozostałych operatorów w danym wyrażeniu arytmetycznym. Patrząc na przykład poniżej, zmienna a najpierw zostanie zwiększona o jeden, a dopiero później jej wartość zostanie przypisana zmiennej b.

Przykład:

int a=0, x=5, b, y;

b = ++a; - preinkrementacja
y = —x; - predekrementacja

Teraz umieścimy operatory za zmienną. W tym przypadku operator zostanie wykonany na końcu. Innymi słowy najpierw wartość zmiennej a zostanie przypisana do zmiennej b, a dopiero później zmienna a zostanie powiększona o jeden.

Dlatego wyniki będą inne niż w poprzednim przykładzie.

b = a++; - postinkrementacja
y = x - -; - postdekrementacja

Modulo

Operacja wyznaczania reszty z dzielenia jednego typu liczbowego przez drugi. W dalszym ciągu napis a mod d = r będzie oznaczał, iż r jest resztą z dzielenia a przez d.

Są różne sposoby określania reszty, a komputery i kalkulatory mają różne sposoby przechowywania i reprezentowania liczb, więc to co dokładnie jest wynikiem operacji modulo zależy od języka programowania i/lub sprzętu.

W niemal każdym systemie komputerowym współczynnik wynikający z dzielenia jest ograniczany do zbioru liczb całkowitych, a reszta r jest zwykle ograniczona przez $0 \leqslant r < |d|$ albo $-|d| < r \leqslant 0$. Wybór między dwiema możliwymi resztami zależy od znaku a lub d oraz użytego języka programowania. Niektóre języki programowania, jak na przykład C89, nawet nie definiują wyniku jeśli zarówno d jak i a jest ujemne.

a mod 0 jest nieokreślone w większości systemów, choć niektóre określają je jako a. Jeśli definicja jest spójna z algorytmem dzielenia, wtedy d = 0 implikuje $0 \leqslant r < 0$, co jest sprzeczne (tzn. zwykła reszta w tym wypadku nie istnieje). Reszta może być wyznaczana równaniami, które korzystają z innych funkcji. Jednym z takich użytecznych równań wyznaczania reszty r jest gdzie oznacza podłogę x.


<C/S>


<C/S>