# **Podstawowe pojęcia**

**Algorytm** – przepis rozwiązania problemu

**Język programowania** – sformalizowany język służący do zapisu algorytmów

**Program** – zapis algorytmu w danym języku programowania

**Programowanie** – proces tworzenia programu, a więc posługiwania się językiem programowania

**Etapy pisania programu:**

1. Sformułowanie i analiza problemu
2. Sporządzenie algorytmu rozwiązania tego problemu
3. Zapisanie algorytmu w pewnym języku programowania
4. Poprawienie ewentualnych błędów składniowych
5. Uruchomienie programu i poprawienie błędów logicznych

**Rodzaje języków programowania:**

* **Języki wysokiego poziomu** – do pisania programów z ich użyciem nie jest konieczna znajomość budowy procesora i komputera
* **Symboliczne (albo asemblerowe)** – do pisania programów z ich użyciem jest konieczna dobra znajomość budowy procesora i komputera

**Cechy języka wysokiego poziomu:**

* Do pisania programów z jego użyciem nie jest konieczna znajomość budowy procesora i komputera
* Jednej instrukcji języka odpowiada po translacji wiele (nawet setki) rozkazów maszynowych (procesora)
* Program wynikowy ma zazwyczaj większą długość i wykonuje się wolniej niż identyczny program napisany w języku symbolicznym
* Użytkownik może dość szybko nauczyć się programować w tym języku
* Tworzenie programu jest szybkie – nie wymaga dużych nakładów czasu

**Cechy języka symbolicznego:**

* Do pisania programów z jego użyciem jest konieczna dobra znajomość budowy procesora i komputera
* Jednej instrukcji języka odpowiada po translacji zazwyczaj jeden rozkaz maszynowy (niekiedy kilka)
* Program wynikowy jest zwarty i wykonuje się szybciej niż identyczny program napisany w języku wysokiego poziomu
* Użytkownik musi poświęcić dużo czasu, by nauczyć się programować w tym języku
* Tworzenie programu jest bardzo czasochłonne

**Translacja –** tłumaczenie programu. Program tłumaczący nazywamy **translatorem**. Rozróżnia się:

* Translatory języków wysokiego poziomu (kompilatory)
* Translatory języków symbolicznych (asemblery)

**Interpretacja** – tłumaczenie programu źródłowego, polegająca na wykonywania programu na podstawie jego postaci źródłowej, proces tłumaczenia programu jest dokonywany krok po kroku, a przetłumaczone fragmenty są wykonywane natychmiast.

**Cechy języka C:**

* Jako język wysokiego poziomu nie wymaga od użytkownika piszącego program znajomości szczegółów wewnętrznej budowy komputera
* Oferuje pełny zakres instrukcji sterujących wykonaniem programu, zawierający instrukcje pętli, warunkowego wykonania i wyboru. Jest to więc język programowania strukturalnego, porównywalny pod tym względem np. z językiem Pascal
* Oferuje duży zestaw operatorów, z których część odpowiada wprost instrukcjom maszynowym, co daje możliwość wykorzystania procesora komputera tak efektywnie, jak w języku asemblerowym
* Dysponuje bogatym zestawem predefiniowanych typów danych i możliwość tworzenia złożonych struktur danych
* Oferuje wskaźniki na dane i funkcje, wraz z arytmetyką umożliwiającą manipulowanie wskaźnikami (adresami pamięci)
* Preprocesor przetwarzający tekst programu, umożliwia m.in. kompilację warunkową i definiowane stałych wartości

**Zalety języka C:**

* **Szybkość** – programy pisane w tym języku działają bardzo szybko dzięki czemu język C może służyć nawet do tworzenia oprogramowania systemowego
* **Uniwersalność** – język C ma kompilatory zarówno dla systemów operacyjnych Windows, jak i Linux. Kod źródłowy napisany zgodnie ze standardami języka, będzie się kompilował zarówno w systemie Windows, jak i Linux
* **Duże możliwości** – dzięki rozbudowanym bibliotekom gotowych funkcji można w tym języku pisać nawet bardzo skomplikowane i duże programy
* **Wspólne cechy z innymi językami** – wiele języków programowania ma składnię bardzo zbliżoną do języka C/C++. Wymienić można tutaj chociażby języki skryptowe takie jak PHP czy JavaScript wykorzystywane powszechnie przy tworzeniu stron WWW. Jeżeli więc ktoś zna język C to dużo łatwiej i szybciej opanuje inne języki programowania.

**Słowa kluczowe** – grupa zarezerwowanych kilkudziesięciu słów angielskich wykorzystywanych w programach jako podstawowy ich składnik

**Instrukcja** – opis czynności jaką należy wykonać na danych, aby je odpowiednio przetworzyć

**Stała** – wartość, która nie zmienia się w trakcie działania programu. Stałą zawsze umieszczamy przed zmienną, która będzie ją wykorzystywała

**Typ zmiennej** – zbiór wartości, jakie może przyjmować zmienna w programie

**Deklaracje** – opisy występujących w programie obiektów, podlegających przetwarzaniu

**Podprogram** – wyodrębniona część programu (ze względu na wielokrotne użycie w różnych miejscach bądź dla zwiększenia czytelności zapisu), posiadająca jednoznaczną nazwę i ustalony sposób wymiany informacji z pozostałymi jego częściami

**Identyfikator** – ciąg znaków, z których pierwszy jest literą, a następne są literami lub cyframi

**Biblioteka** – zbiór funkcji, klas i innych konstrukcji programistycznych, z których korzystają programiści piszący program

**Konsolidator (linker)** – jeden z programów składowych kompilatora. Konsolidator łączy biblioteki z plikami zawierającymi kody źródłowe, które korzystają z dołączanych bibliotek

**Deklaracja zmiennej –** operacja zgłoszenia kompilatorowi, że programista będzie używał zmiennej o określonej nazwie, wobec tego kompilator musi dla tej zmiennej zarezerwować odpowiednią ilość bajtów w pamięci RAM. Dokładna ilość rezerwowanej pamięci zależy od typu zmiennych prostych jest to 1-8 B.

|  |  |  |
| --- | --- | --- |
| Nazwa typu | Ilość bajtów | Zakres wartości |
| bool | 1 | **false** lub **true** |
| char | 1 | od -128 do 127 |
| unsigned char | 1 | od 0 do 255 |
| short | 2 | od -32768 do 32767 |
| unsigned short | 2 | od 0 do 65535 |
| int | 4 | od -2147483648 do 2147483647 |
| unsigned int | 4 | od 0 do 4294967295 |
| long | 4 | od -2147483648 do 2147483647 |
| unsigned long | 4 | od 0 do 4294967295 |
| long long | 8 | od -9223372036854775808 do 9223372036854775807 |
| unsigned long long | 8 | od 0 do 18446744073709551615 |
| float | 4 | 3.4E +/- 38 (7 cyfr) |
| double | 8 | 1.7E +/- 308 (15 cyfr) |
| long double | 8 | 1.7E +/- 308 (15 cyfr) |

**Komentarz** – fragment programu źródłowego, którego jedynym celem jest informowanie o czymś osoby go czytającej – nie ma on żadnego wpływu na program i jest przez kompilator pomijany. W języku C++ **stosuje się dwa rodzaje komentarzy:**

* Jednolinijkowy zaczynający się znakami //
* Wielolinijkowy (blokowy) w postaci: /\* tekst \*/

**Zasady pisania kodu źródłowego:**

* Jedna instrukcja w jednej linii
* Stosowanie wcięć
* Dodawanie komentarzy
* Nadawanie zmiennym znaczących nazw
* Podział programu na mniejsze fragmenty – podprogramy

# **Biblioteki**

|  |  |
| --- | --- |
| Nazwa biblioteki | Funkcje |
| <iostream> | Systemowe |
| <math.h> | abs(), asin(), cos(), exp(), log(), pow(), round(), sin(), sqrt(), floor(), ceil() |
| <stdlib.h> | system(), rand(), srand(), |
| <time.h> | time() |
| <conio.h> | getch(), clrscr() |
| <windows.h> | Definiuje typy danych |
| <ctime> | Obliczanie czasu |

# **Operatory**

**Operatory arytmetyczne** – stosuje się do obliczania wartości liczbowych**:**

* **+** (dodawanie)
* **-** (odejmowanie)
* **\*** (mnożenie)
* **/** (dzielenie)
* **%** (reszta z dzielenia całkowitego)

**Operatory logiczne** – służą do wykonywania operacji logicznych na wartościach typu*bool* oraz na wartościach całkowitych**:**

* **!** (negacja)
* **&&** (koniunkcja)
* **||** (alternatywa)

**Operatory relacyjne** – służą do konstrukcji wyrażeń porównania, wynikiem wyrażenia porównania jest typu *bool* i posiada wartość *true* lub *false*:

* **==** (równy)
* **!=** (różny)
* **<** (mniejszy)
* **>** (większy)
* **<=** (nie większy)
* **>=** (nie mniejszy)

**Operatory inkrementacji i dekrementacji** – powodują odpowiednio zwiększenie lub zmniejszenie wartości argumentu o jeden:

* **++ inkrementacja** – zwiększenie o jeden
* **-- dekrementacja** – zmniejszenie o jeden

**Operator przypisania** – służy do nadawania zmiennym nowych wartości, jest stosowany we wszelkiego rodzaju wzorach, zapisywany za pomocą znaku równości

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Arytmetyczne | Logiczne | Relacje | Inkrementacji i dekrementacji | Przypisania |
| + dodawanie | ! negacja | == równy | ++ zwiększenie o jeden | = nadanie wartości |
| - odejmowanie | && koniunkcja | != różny | -- zmniejszenie o jeden |  |
| \* mnożenie | || alternatywa | < mniejszy |  |  |
| / dzielenie |  | > większy |  |  |
| % reszta z dzielenia całkowitego |  | <= nie większy |  |  |
|  |  | >= nie mniejszy |  |  |

Po dodaniu biblioteki ***math.h***można wykonywać skomplikowane wyrażenia, tj:

|  |  |  |  |
| --- | --- | --- | --- |
| Funkcja | Znaczenie | Przykład | Wartość |
| abs(x) | Wartość bezwzględna z x | abs(-10) | 10 |
| asin(x) | Arcus sinus z x | asin(0.8415) | 1.00005 |
| cos(x) | Cosinus z x | cos(0) | 1 |
| exp(x) | e do potęgi x | exp(1) | 2.71828 |
| log(x) | Logarytm naturalny z x | log(10) | 2.30259 |
| pow(x,n) | Oblicza x do potęgi n | pow(2,3) | 8 |
| round(x) | Zaokrągla x do najbliższej liczby całkowitej | round(2.71828) | 3 |
| sin(x) | Sinus z x | sin(1) | 0.841471 |
| sqrt(x) | Pierwiastek kwadratowy | sqrt(16) | 4 |
| floor(x) | Zaokrąglenie w dół | floor(3.2) | 3 |
| ceil(x) | Zaokrąglenie w górę | ceil(3.2) | 4 |

**Instrukcja** w języku programowania to opis czynności, jaką należy wykonać na danych, aby je odpowiednio przetworzyć. Instrukcja może zawierać wewnętrzne komponenty (np. wyrażenia).

**Wszystkie instrukcje dzielą się na:**

* **Proste** – np. przypisania, skoku, powrotu
* **Złożone** – warunkowe, iteracyjne (pętle)

Jedną z najczęściej wykorzystywanych instrukcji języka C++ są **instrukcje warunkowe**, które uzależniają wykonanie innych instrukcji od spełnienia określonego warunku. Zastępują one bloki decyzyjne ze schematu blokowego.

# **Instrukcje warunkowe**

**Do instrukcji warunkowych zaliczamy:**

**if –** wykonuje instrukcje, jeżeli warunek jest prawdziwy, jeżeli nie – wykonuje instrukcje zapisane po else

|  |  |
| --- | --- |
| **if** (warunek)  {  instrukcja1  }  **else**  **{**  instrukcja2  } | **if** (a==1)  {  cout << ”Dobrze!” << endl;  }  **else**  {  cout << “Źle!” << endl;  } |

**switch … case** – stosowana wtedy, gdy warunek w programie jest tak sformułowany, że istnieje kilka różnych wariantów rozgałęzienia (a nie tylko dwa); wykonanie kodu po słowie kluczowym *default* następuje tylko wtedy, gdy żadna z opcji po słowie *case* nie zostanie wykorzystana

|  |  |
| --- | --- |
| **switch** (wyrażenie)  {  **case x1:** ciąg instrukcji;  **case x2:** ciąg instrukcji;  **case x3:** ciąg instrukcji;  …  **case xn:** ciąg instrukcji;  **default:** ciąg instrukcji;  } | **switch** (wybór)  {  case 1**:** cout << “1” << endl;  break;  case 2**:** cout << “2” << endl;  break;  default**:** cout << “Źle!” << endl;  break;  } |

# **Instrukcje iteracyjne**

Drugą grupę ważnych instrukcji w języku C++ stanowią **instrukcje iteracyjne**, zwane też **pętlami** lub **pętlami programowymi**. Działanie instrukcji tego typu polega na przetwarzaniu ciągłych innych instrukcji określoną ilość razy. W języku C++ istnieją aż trzy instrukcje iteracyjne:

**for** – bardzo często używana pętla, stosuje się ją najczęściej wtedy, gdy z góry wiemy, ile razy ma być powtórzona, sama zwiększa zmienną sterująca pętli

|  |  |
| --- | --- |
| **for** (inicjacja; warunek; krok)  {  powtarzane instrukcje  } | **for** (i=1,i<=10;i++)  {  cout << “i= “ << i << endl;  cout << endl;  } |

**while** – pętla wykonuje się tak długo, jak długo prawdziwy jest warunek po słowie while, gdy warunek staje się fałszywy - pętla kończy działanie; pętla może nie wykonać się ani razu, gdy warunek od początku jest fałszywy, sama nie zwiększa zmiennej sterującej pętli

|  |  |
| --- | --- |
| **while** (warunek)  {  powtarzane instrukcje  } | i=1;  **while** (i<=2)  {  cout << i << endl;  i=i+0.1  } |

**do … while** – pętla wykonuje się tak długo, jak długo prawdziwy jest warunek po słowie while, gdy warunek staje się fałszywy – pętla kończy działanie; pętla musi wykonać się co najmniej raz, gdyż warunek sprawdzany jest dopiero po wykonaniu instrukcji, sama nie zwiększa zmiennej sterującej pętli

|  |  |
| --- | --- |
| **do**  {  powtarzane instrukcje  }  **while** (warunek) | i=1;  **do**  {  cout << ”i” << endl;  i=i+0.1;  }  **while** (i<=2) |

**Ta sama pętla rożnymi metodami:**

|  |  |
| --- | --- |
| For | **for** (x = 0; x < 10; x++)  {  cout<<x<<endl;  } |
| While | **while** (x < 10)  {  cout<<x<<endl;  x++;  } |
| Do … while | **do**  {  cout<<x<<endl;  }  **while** (x < 10); |

# **Podprogramy**

**Zasady tworzenia zmiennych wykorzystywanych w programie:**

* Nazwa zmiennej może składać się wyłącznie z liter, cyfr i znaków podkreślenia
* Pierwszym znakiem nazwy zmiennej musi być wielka lub mała litera
* Nie można używać liter z ogonkami i akcentami
* Nie można używać spacji

**Podprogram** – wyodrębniona cześć programu, która realizuje fragment algorytmu

**Rodzaje podprogramów:**

* **Procedury** – wykonanie pewnej sekwencji czynności (np. czyszczenie ekranu, narysowanie pewnych elementów, wygenerowanie dźwięków)
* **Funkcje** – obliczanie jednej wartości

**Korzyści wynikające ze stosowania podprogramów:**

* **Rozkład problemu na mniejsze i prostsze składniki** – rozbicie problemu na mniejsze podproblemy i rozwiązanie każdego z nich osobno jest naturalną techniką radzenia sobie ze złożonymi problemami programistycznymi
* **Zwiększenie czytelności i przejrzystości programów** – podzielenie programu na szereg mniejszych fragmentów ułatwia jego pisanie, uruchamianie i ewentualne modyfikowanie
* **Unikanie powtórzeń** – jeśli pewien ciąg instrukcji powtarza się kilkakrotnie w programie, wygodnie jest wyodrębnić go w postaci podprogramu a następnie wywołać go kilka razy dzięki temu nieraz znacznie skraca się program źródłowy

**W języku C występują dwa rodzaje funkcji:**

* **Funkcje typu void** – funkcja tego typu nie zwraca żadnych danych, które można byłoby poddać dalszej obróbce. Funkcje takie mogą wykonywać pewne czynności, ale nie przekazują informacji zwrotnej
* **Funkcje zwykłe** – czyli funkcje, które zwracają jakąś wartość

# **Tablice**

**Tablica** – kontener uporządkowanych danych takiego samego typu, w którym poszczególne elementy dostępne są za pomocą indeksowania. Rozmiar tablicy jest albo ustalany z góry (tablice statyczne) albo może zmieniać się w trakcie wykonywania programu (tablice dynamiczne). Tablice jednowymiarowe mogą przechowywać inne tablice, dzięki czemu uzyskuje się tablice wielowymiarowe. W praktyce tablic używa się do przechowywania większych ilości zmiennych tego samego typu

**Przy tworzeniu tablic należy pamiętać o:**

* Typie elementów które będzie przechowywać tablica
* Nazwie tablicy
* Liczbie elementów (wielkość tablicy)

**Należy pamiętać, że numerowanie elementów tablic w języku C++ zaczyna się od zera.**

# **Metody zwiększanie efektywności pracy programów**

**Zasady pozwalające unikać części błędów, łatwiejsze ich wykrywanie i usuwanie, zwiększenie szybkości pracy programu:**

* Podczas obliczeń należy stosować liczby całkowite, gdyż obliczenia na liczbach rzeczywistych wykonywane są wolniej
* Główny algorytm należy rozbić na mniejsze problemy i każdy z nich zrealizować za pomocą osobnego podprogramu
* Stosowanie zasad zwiększających czytelność kodu źródłowego
* W wyrażeniach używanych w programie należy stosować typy operacji, które procesor wykonuje szybciej
* Jeżeli program ma dużo liczyć należy ograniczyć ilość złożonych funkcji matematycznych
* Jeżeli jedna funkcja będzie wykorzystywane wielokrotnie należy zdefiniować jest wartość jako stałą
* Należy starannie przygotowywać algorytm
* Gotowy program należy przetestować wprowadzając przykładowe dane i sprawdzając czy uzyskany wynik jest zgodny z przewidywanymi, należy też podać błędne dane by zobaczyć jak program na to zareaguje
* Jeżeli tworzy się dłuższy program to należy powtarzające się fragmenty kodu (podprogramy) pogrupować w biblioteki.

# **Obsługa plików**

**Plik** jest zbiorem elementów tego samego typu, tyle że ich liczba jest zmienna. Jest ona zależna od przebiegu wykonanego programu (który z tego pliku korzysta) oraz od skojarzenia pliku z fizycznym zbiorem danych

Od tablicy plik różni się zmienną liczbą elementów oraz metodą dostępu do poszczególnych elementów, jest to dostęp sekwencyjny (w dowolnym momencie może być dostępny tylko jeden element pliku, a dostęp do pozostałych elementów jest możliwy dopiero po wykonaniu pewnych operacji na pliku)

**Podczas przetwarzania plików można wyróżnić następujące sytuacje:**

* **Zakładanie** – tworzenie nowego fizycznego zbioru danych
* **Wprowadzanie** do nowo założonego pliku elementów
* **Dodawanie** nowych elementów do pliku już istniejącego
* **Przeszukiwanie plików** – odnajdywanie w nim określonych elementów
* **Odczyt** elementów z pliku
* **Wymiana** elementów pliku

**Ogólny schemat przetwarzania pliku:**

* Opis zmiennej plikowej
* Skojarzenie pliku z fizycznym zbiorem danych
* Otwarcie pliku
* Wykonanie operacji na pliku
* Zamknięcie pliku

**W programie wygląda to w następujący sposób:**

|  |  |
| --- | --- |
| fstream plik;  plik.open(„dane.txt”,ios::out);  plik<<linia<<endl;  plik.close() | Opis zmiennej plikowej  Skojarzenie zmiennej plikowej z fizycznym plikiem „dane.txt”  Wykonanie operacji na pliku (zapis)  Zamknięcie pliku |

# **Iteracja i rekurencja**

**Iteracja** – technika programistyczna polegająca na wielokrotnym powtarzaniu tej samej instrukcji lub grupy instrukcji. Iteracja jest więc pętlą programową pozwalającą na wielokrotne wykorzystanie tych samych instrukcji. Jest to bardzo ważna metoda, gdyż umożliwia wykorzystanie głównej zalety komputera – szybkiego wykonywania obliczeń. Wiele problemów dzięki temu można rozwiązać metodą siłową. Jak ważna jest to metoda świadczy fakt, że w wielu jeżykach programowania np. C++ istnieją 3 instrukcje iteracyjne **for**, **while**, **do while**.

**Rekurencja (rekursja)** – metoda programistyczna polegająca na odwoływaniu się funkcji do samej siebie. Istnieje grupa problemów które można rozwiązać metodą rekurencyjną np. obliczanie potęgi całkowitej, obliczanie silni czy też ciągów Fibonacciego (ciąg Fibonacciego to ciąg liczb naturalnych którego wyrazy liczymy ze wzoru sekwencyjnego)

# **Struktury**

**Struktura** to złożony typ danych pozwalający przechować różne informacje. Za pomocą struktur możliwe jest grupowanie wielu zmiennych o różnych typach w jeden twór. Składowanymi elementami struktur są pola mające swoje unikatowe nazwy. W innych językach programowania struktury mogą nosić nazwy rekordów. Struktury umożliwiają programistom tworzenie baz danych.

Elementami składowymi struktury są pola, które mają własne unikatowe nazwy. Poprzez podanie nazwy otrzymuje się dostęp do danego pola.

Struktury są powszechnie wykorzystywane w programowaniu, pozwalając bowiem na przejrzysty sposób opisywać złożone obiekty.

Przykładem struktury może być informacja o książce, której pola będą zawierały: imię i nazwisko autora (typ łańcuchowy), tytuł (typ łańcuchowy), rok wydania (typ całkowity), liczba stron (typ całkowity), nazwa wydawnictwa (typ łańcuchowy).

**Ogólna postać deklaracji i przykładowa deklaracja:**

|  |  |
| --- | --- |
| **struct** nazwa {  typ nazwa\_pierwszego\_elementu;  typ nazwa\_drugiego algorytmu;  typ nazwa trzeciego\_elementu;  …  }; | **struct** auto {  string marka;  string model  int rok\_produkcji;  float pojemność;  }; |

**Jeśli struktura zostanie zdefiniowana przed funkcją main()**, będzie ona widoczna dla wszystkich bloków programu – **będzie globalna**. Słowo kluczowe struct informuje kompilator, że zostanie zadeklarowana struktura. Po stworzeniu struktury dostępny jest nowy typ – w tym przypadku auto. Można teraz tworzyć zmienne na jego podstawie. Nie różni się to niczym od tworzenia zmiennych opartych na typach standardowych takich jak int, char, float, itp. Wygląda to następująco:

|  |
| --- |
| auto osobowe;  osobowy.marka=„Renault”;  osobowy.model=„Megane”;  osobowy.rok\_produkcji=2015; |

Zadeklarowana zmienna osobowy jest strukturą o polach takich jak struktura auto. Aby wstawić dane do kolejnych poł trzeba dokonać podstawienia: nazwa\_zmiennej-kropka-nazwa\_pola=wartość, ilustruje to powyższy przykład. W ten sposób można wstawić dane do pól struktury albo wyświetlać zawartość kolejnych pól.

# **Algorytmy**

Jednym z etapów tworzenia programu komputerowego jest opracowanie rozwiązującego go algorytmu w wypadku nieskomplikowanych problemów można etap tworzenia algorytmu pominąć, gdyż każdy z umiejętnościami zna wiele prostych i standardowych algorytmów.

W przypadku bardziej złożonych problemów które np. realizowane będą przez wielu programistów należy przygotować (na papierze lub w edytorze) algorytm realizujący ten złożony problem.

**Istnieje wiele metod zapisu algorytmów. Najbardziej znane metody to:**

* **Lista kroków** – to metoda zapisu algorytmów polegająca na wypisywaniu (na papierze lub w edytorze) kolejnych kroków pozwalających rozwijać dany problem np.:
  + Przedstawić algorytm rozwiązywania układów dwóch równań liniowych z dwoma niewiadomymi metodą listy kroków.
    - Rozwiązanie:
    - Wyznaczyć „y” z pierwszego równania
    - Podstawić za „y” do równania drugiego
    - Z równania drugiego wyznaczyć „x”
    - Wrócić do podstawienia na „y” i obliczyć go
  + Na powyższym przykładzie widać, że ten sposób zapisu algorytmów jest mało czytelny oraz trudny do analizy i przeróbek.
* **Schemat blokowy** – to metoda zapisu algorytmów w postaci graficznej. Schemat blokowy składa się z bloków o różnych kształtach oraz łączących je linii ze strzałkami. Tak zapisany algorytm jest czytelny i łatwy do analiz i przeróbek.

**Zalety schematów blokowych**:

* Czytelność i przejrzystość
* Łatwość analizy i wprowadzania poprawek
* Możliwość przeprowadzenia symulacji pracy algorytmu, a więc jego poprawności

**Wyróżnia się dwa rodzaje zapisu algorytmów:**

* **Algorytm liniowy** – to taki algorytm, który realizuje jeden ciąg zadań, czynności w nim zapisane są wykonywane sekwencyjnie (po kolei) tak jak zostały zapisane, w algorytmie liniowym nie sprawdza się warunków (nie ma bloków decyzyjnych)
* **Algorytm rozgałęziony** (z warunkami) – to algorytm, który zawiera instrukcje warunkową (blok decyzyjny). W algorytmie z warunkami może wystąpić kilka alternatywnych ciągów zadań. Wybór jednego z nich następuje w zależności od spełnienia lub nieokreślonego warunku

**Schematy blokowe są najoporniejszą metodą zapisu algorytmów.**

**Wyróżnia się kilka podstawowych kształtów bloków:**

|  |  |
| --- | --- |
| Kształt | Nazwa |
| Początek | Blok początku programu |
| Koniec | Blok końca programu |
| a=4 b=3a+1 | Blok przetwarzania |
| k>0  N  T | Blok decyzyjny |
| Czytaj a, b | Blok współpracy z urządzaniami wejścia-wyjścia |
| A | Blok łączący |

**Bloki:**

* **Blok początku i końca programu** – występują tylko raz i rozpoczynają i kończą każdy algorytm
* **Blok przetwarzania** – umieszczany w miejscach w których znajduje się operacja lub grupa operacji
* **Blok decyzyjny** – umieszczany w miejscach w których komputer podejmie decyzje dotyczące toku dalszych obliczeń, a więc wybiera do którego bloku ma przejść jako do następnego, wewnątrz umieszcza się warunek, który ma być badany
* **Blok współpracy z urządzeniami wejścia-wyjścia** – umieszczany w miejscach w których dane są wprowadzane do komputera lub wyprowadzane, jeżeli dane są wyczytywane blok zawiera słowo CZYTAJ, jeśli wyprowadzane zawiera słowo DRUKUJ
* **Blok łączący** – stosowany, kiedy schemat jest za duży, aby zmieścić go na jednej stronie albo kiedy ciężko poprowadzić strzałki łączące dwa bloki

# **Metoda dziel i zwyciężaj**

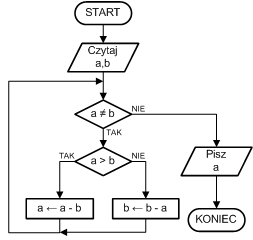
Dziel i zwyciężaj jest jedną z głównych metod projektowania algorytmów w informatyce, prowadzącą do bardzo efektywnych rozwiązań, w metodzie tej problem dzieli się rekurencyjnie na 2 lub więcej mniejszych problemów tego samego typu lub podobnego, tak długo aż fragmenty staną się wystarczająco proste do bezpośredniego rozwiązania. Następnie rozwiązania otrzymane dla podproblemów scala się uzyskując rozwiązanie całego zadania.

**Z metody tej korzysta wiele popularnych algorytmów np.:**

* Sortowanie przez scalanie
* Sortowanie szybkie
* Wyszukiwanie binarne

**Wyszukiwanie binarne** to algorytm opierający się na metodzie dziel i zwyciężaj, który w czasie logarytmicznym stwierdza czy szukany element znajduje się w uporządkowanej tablicy, a jeśli tak to podaje jego indeks (numer). Przykładowo, jeśli zawiera milion elementów wyszukiwanie binarne musi sprawdzić 20 elementów (gdyż ), w celu scalenia żądanej wartości, dla porównania wyszukiwanie liniowe wymaga w najgorszym wypadku sprawdzenia wszystkich elementów.

**Algorytm Euklidesa** – to algorytm znajdowania największego wspólnego dzielnika dwóch liczb



**NWD (największy wspólny dzielnik)** – dla dwóch liczb całkowitych to największa liczba naturalna dzieląca każdą z nich

**NWW (najmniejsza wspólna wielokrotność)** – dwóch liczb naturalnych to najmniejsza liczba naturalna której dzielnikiem są te cyfry

**Między NWD a NWW zachodzi zależność:**

# **Kryptografia**

**Kryptografia** to dziedzina wiedzy o przekazywaniu informacji w sposób zabezpieczony przed niepowołanym dostępem. Kryptologie można podzielić na dwie kategorie:

* **Kryptografia** – gałąź wiedzy o przekazywaniu wiadomości
* **Kryptoanaliza** – gałąź wiedzy o przełamywaniu zabezpieczeń oraz o deszyfrowaniu wiadomości bez klucza

W społeczeństwie technologicznym kryptologia znajduje zastosowanie w rozwiązaniach zapewniających bezpieczeństwo kart bankomatowych, haseł komputerowych czy też handlu elektronicznego. Jednak najważniejszym jej zadaniem jest wykorzystanie w dyplomacji i wojskowości.

**Szyfrowanie** – zamiana tekstu jawnego w szyfrowany

**Deszyfrowanie** – zamiana tekstu szyfrowanego na jawny

**Szyfr** – jest to para algorytmów służąca do przeprowadzania szyfrowania i deszyfrowania. Wraz z algorytmami dodatkowo używa się kluczy, czyli pewnych parametrów, od których zależy wynik szyfrowania i deszyfrowania. Znajomość algorytmu i szyfrogramu bez dostępu do klucza nie pozwoli na odtworzenie tekstu jawnego

**Klucz** – pewne parametry, od których zależy wynik szyfrowania i deszyfrowania

**Wyróżnia się dwa rodzaje kryptografii:**

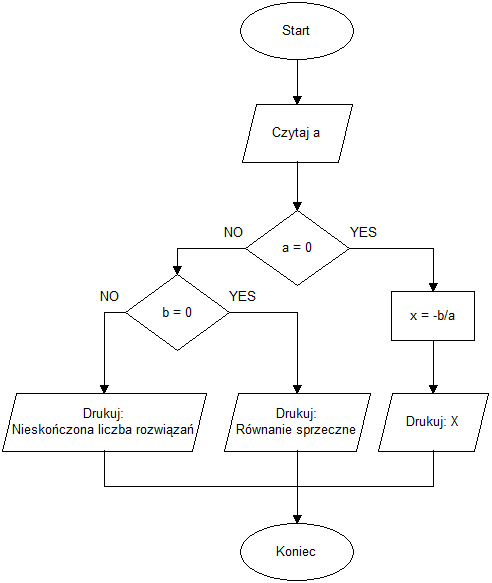
* **Kryptografia symetryczna** – w wypadku jej użycia nadawca i odbiorca wiadomości używają tego samego klucza
* **Kryptografia asymetryczna (klucza publicznego)** – w wypadku jej użycia stosuje się dwa klucze: klucz publiczny które może być swobodnie rozpowszechniany oraz klucz prywatny, służący do deszyfrowania wiadomości który musi być przechowywany w sekrecie

# **Algorytm z warunkami zagnieżdżonymi**

**Cechy algorytmu z warunkami zagnieżdżonymi:**

* Składa się z kilku warunków
* Po sprawdzeniu jednego warunku trzeba sprawdzić kolejne
* W schemacie blokowym wychodząc z jednego bloku decyzyjnego trafiamy do kolejnego

**Przykładem algorytmu z warunkami zagnieżdżonymi** może być algorytm rozwiązania równania liniowego ax+b=0



# **Algorytm iteracyjny**

Algorytm iteracyjny to algorytm, którego wynik uzyskuje się poprzez iterację, czyli powtórzenie danej operacji z góry określoną liczbę razy

# **Pętle zagnieżdżone**

Zagnieżdżanie pętli podobnie jak instrukcji warunkowej polega na wywołaniu jednej pętli wewnątrz drugiej oznacza to, że na jedną iterację pętli zewnętrznej zostanie wykonany cały przebieg pętli wewnętrznej. Instrukcje iteracyjne w C++ można zagnieżdżać dowolnie. Im więcej zagnieżdżeń tym większa złożoność obliczeniowa algorytmu

# **Zasięg zmiennej**

**Zmienne występujące w programie można podzielić na:**

* **Zmienne globalne** – to takie zmienne które są dostępne w całym programie i przez cały czas jego działania. Zmienne globalne deklaruje się pomiędzy nagłówkiem a funkcją main. Ponieważ zmienne globalne są widoczne zatem są również widoczne wewnątrz funkcji main. Nazwy zmiennych globalnych muszą być między sobą różne ???
* **Zmienne lokalne** – to takie zmienne które są dostępne tylko w pewnej części programu, zazwyczaj tylko w pewnej chwili działania programu a nie przez cały czas. Zmienne lokalne mogą być utworzone w dowolnym miejscu danej funkcji, czyli również wewnątrz funkcji main. Można je tworzyć również wewnątrz bloku programu np. wewnątrz pętli

# **Definiowanie podprogramów**

Przy definiowaniu podprogramów wykorzystuje się tam **parametry formalne**, są to wprowadzone w odpowiedni sposób nazwy zastępujące właściwe obiekty używane potem przy odwoływaniu się do podprogramu. Parametry formalne umożliwiają zapisanie podprogramu w postaci ciągu instrukcji do zapisu algorytmu. Przy wywołaniu tego podprogramu parametry formalne są zastępowane obiektami rzeczywistymi z poza podprogramu czyli tzw. parametrami aktualnymi. Jeżeli jedna funkcja (np. min) wywołuje drugą funkcję to muszą one między sobą przekazywać parametry. W języku C++ parametry można przekazywać na trzy sposoby:

* Przez **wartość**
* Przez **referencję**
* Przez **wskaźnik**

Najpopularniejszą metodą jest przekazywanie parametrów przez wartość. W momencie wykonania funkcji, tworzona jest na jej potrzeby zmienna lokalna o podanej nazwie i do niej jest kopiowana nowa wartość. Po zakończeniu działanie funkcji wszystkie zmienne powiązane z parametrami przekazywanymi do niej przestają istnieć. ???

Po wyjściu z funkcji odwołujemy się do oryginalnej zmiennej, która nie została zmodyfikowana.

# **Złożoność obliczeniowa**

**Złożoność algorytmu** – ilość zasobów systemowych niezbędnych do wykonania algorytmu

**W zależności od rozważanego zasobu mówimy o:**

* Złożoności **czasowej**
* Złożoności **pamięciowej**

Oczywiście w większości wypadków ilość potrzebnych zasobów będzie się różnić w zależności od danych wejściowych z zakresu danego zagadnienia

# **Kompresja**

Kompresja danych polega na zmianie sposobu zapisu danych tak, aby zmniejszyć nadmiarowość i tym samym objętość zbioru. Innymi słowy chodzi o wyrażenie tego samego zestawu danych, za pomocą mniejszej liczby bitów. Działaniem przeciwnym do kompresji jest dekompresja.

**Kompresja dzieli się na:**

* **Bezstratną** - w której z postaci skompresowanej danych można odzyskać identyczną postać pierwotną
* **Stratną** - w której odzyskanie postaci pierwotnej danych jest niemożliwe, jednak główne właściwości danych zostają zachowane

Nie można odzyskać pierwotnej postaci skompresowanych danych gdyż w trakcie kompresji część danych jest bezpowrotnie odrzucana (tracona).

**Algorytmy kompresji dzieli się na:**

* Algorytmy **zastosowania ogólnego**
* Algorytmy **do określonego typu danych**

Z definicji nie istnieją algorytmy kompresji stratnej zastosowania ogólnego, ponieważ dla różnych typów danych konieczne jest zachowanie różnych właściwości.

Większość algorytmów bezstratnych to algorytmy zastosowania ogólnego a ich drobne przeróbki lepiej działają z określonymi typami danych.

Kompresję stratną można stosować tylko do niektórych typów danych, np. do zdjęć, dźwięku. Do innych typów danych, np. programy, rysunki, teksty, itp. można stosować tylko kompresję bezstratną.

Do niektórych typów danych (zdjęcia, dźwięk) można stosować kompresję zarówno stratną jak i bezstratną, np.:

* **Do dźwięku:**
  + **mp3** - kompresja stratna
  + **flac** - kompresja bezstratna
* **Do zdjęć:**
  + **jpg** - kompresja stratna
  + **png** - kompresja bezstratna

**Kompresja stratna** może zmniejszyć wielkość oryginalnego pliku nawet o 99% ale kosztem jakości. Im większy współczynnik kompresji tym mniejszy plik ale gorsza jakość zdekompresowanych danych (zdjęcia, dźwięku).

W wypadku użycia kompresji bezstratnej wielkość pliku wyjściowego zmniejsza się niedużo 10-40% ale po dekompresji odzyskuje się oryginalny plik wyjściowy.

Do kompresji własnych danych użytkownik komputera może wykorzystać wiele znanych programów, takich jak np.:

* **WinZip**
* **WinRar**
* **7-zip –** archiwizer rozpowszechniany na licencji LGPL. Wykorzystuje on swój własny format kompresji - 7z, który charakteryzuje się wyjątkowo dużym stopniem kompresji. Obsługuje również znane formaty z innych programów archiwizujących: ZIP, RAR

# **Systemy plików**

**System plików** – to zbiór zasad i metod dotyczących przechowywania plików, zarządzania nimi, ich nazewnictwa oraz przechowywania dodatkowych informacji o nich. Innymi słowy jest on sposobem organizacji plików na dysku. System plików umożliwia wykonywanie następujących zadań:

* Tworzenie i usuwanie pliku
* Otwieranie i zamykanie pliku
* Odczyt i zapis danych do pliku
* Zmiana nazwy pliku
* Tworzenie i usuwanie folderu

**Systemy plików tworzone są podczas formatowania dysków i są obecne na wszystkich nośnikach danych takich jak:**

* Dyskietki
* Dyski twarde
* Dyski SSD
* Pendrive
* Karty pamięci flash
* itd.

**Większość systemów operacyjnych posiada własny** (macierzysty) system plików, rozwijany równolegle z nim (np. FAT w DOS-ie, NTFS w Windows NT/7/8/10 czy też ext/ext2/ext3/ext4 w Linuxie). Same systemy operacyjne potrafią obsługiwać wiele różnych systemów plików.

**Formatując jakikolwiek nośnik pamięci zawsze trzeba wybrać odpowiedni typ systemu plików. Najpopularniejsze systemy plików to:**

* **FAT** (FAT16) – to system plików powstały w latach siedemdziesiątych i wykorzystywany głównie w systemie DOS i pierwszych wersjach Windows. Skrót FAT pochodzi od File Allocation Table. System plików FAT jest oparty na tablicy, która opisuje, w którym klasterze dysku system operacyjny powinien szukać zapisanego w nim pliku.
* **FAT32** – to następca systemu FAT, pojawił się w Windows 95 OSR2.  Rozmiar partycji obsługiwanej w tym systemie plików nie może przekroczyć 2 TB. Jest używany do tej pory, głównie na zewnętrznych nośnikach danych (dyski przenośne, pendrive). Jego poważnym mankamentem jest to, że maksymalny rozmiar pliku w tym systemie to 4 GB. Natomiast podstawową zaletą tego systemu jest kompatybilność z szeroką gamą urządzeń i systemów operacyjnych. FAT32 jest obsługiwany przez system Windows, Mac OS X, Linux, konsole do gier i praktycznie wszystkie inne urządzenia z portem USB
* **NTFS** – to system plików, który pojawił się w systemie operacyjnym Windows NT i jest wykorzystywany przez jego następców (Windows 7/8/10). Partycja, na której zainstalowany jest Windows, musi być sformatowana w NTFS. Pozwala on obsługiwać dyski o dużej pojemności a także znosi ograniczenie wielkości pliku do 4GB. Został zaprojektowany pod kątem pracy z wieloma użytkownikami. Wprowadza nowe możliwości takie jak:
  + prawa dostępu do plików i katalogów
  + system zarządzania zapisem i odczytem danych
  + księgowanie czyli mechanizm, który zwiększa bezpieczeństwo systemu
  + szyfrowanie plików i katalogów
  + kompresja danych „w locie”
* **ext4** – to czwarta wersja rozszerzonego systemu plików czyli najpopularniejszego systemu plików dla Linuxa. Jest obecny na rynku od 2008 roku. To nowoczesny, stabilny oraz dobrze przetestowany system plików oparty na systemie ext3. Oferuje wysoką wydajność dzięki przeciwdziałaniu fragmentacji plików. Dostępny w wielu dystrybucjach Linuxa.
* **exFAT** – to system plików, który został stworzony przez firmę Microsoft do obsługi przenośnych nośników danych takich jak: zewnętrzne dyski 2,5”, pendrive, karty pamięci flash. Jest obsługiwany przez systemy operacyjne od Windows 7 wzwyż. exFAT wykorzystywany jest zamiast systemu FAT32 czy też NTFS (który to system nie jest najlepszym rozwiązaniem dla przenośnych nośników danych). Zalety sytemu exFAT w porównaniu do FAT32 to:
  + obsługa dysków o bardzo dużych rozmiarach
  + praktycznie nieograniczona wielkość pojedynczego pliku (16 EB)
  + nieograniczona liczba plików w pojedynczym katalogu
  + wprowadzony system zarządzania prawami własności
  + udoskonalona wydajność przy kopiowaniu/usuwaniu plików

Podsumowując można stwierdzić, że nie ma idealnego systemu plików, który sprawdziłby się we wszystkich zastosowaniach. W zależności od potrzeb stosuje się różne z nich. NTFS jest najlepszym systemem plików dla dysków wewnętrznych/systemowych, natomiast exFAT dla pamięci przenośnych flash. Niekiedy jednak te ostatnie trzeba sformatować w systemie FAT32, by były kompatybilne ze starszymi urządzeniami. Obecnie FAT32 jest idealnym wyborem dla pendrive'ów o niewielkich pojemnościach. Dzięki temu systemowi plików mamy pewność, że nośnik zadziała na niemal każdym urządzeniu