Marek Gągolewski
INSTYTUT BADAŃ SYSTEMOWYCH PAN
WYDZIAŁ MATEMATYKI I NAUK INFORMACYJNYCH POLITECHNIKI WARSZAWSKIEJ

Algorytmy
i podstawy programowania

1. Etapy tworzenia oprogramowania. Algorytm

Materiały dydaktyczne dla studentów matematyki
na Wydziale Matematyki i Nauk Informacyjnych Politechniki Warszawskiej
Ostatnia aktualizacja: 1 października 2016 r.

Spis treści

1.1. Wprowadzenie

Science is what we understand well enough to explain to a computer.
Art is everything else we do.
D. E. Knuth

Przygodę z algorytmami i programowaniem czas zacząć! W trakcie naszych zajęć poznasz wiele nowych, ciekawych zagadnień. Celem, który chcemy pomóc Ci osiągnąć, jest nie tylko bliższe poznanie komputera (jak wiadomo, bliskość sprzyja nawiązywaniu przyjaźni, często na całe życie), ale i rozwijanie umiejętności rozwiązywania ważnych problemów, także matematycznych.

Na pewno poważnie myślisz o karierze matematyka (być może w tzw. sektorze komercyjnym lub nawet naukowo-badawczym). Pamiętaj o tym, że opanowanie umiejętności programowania komputerów nie będzie wcale atutem w Twoim życiu zawodowym. Będzie warunkiem koniecznym powodzenia! Komputery bowiem towarzyszą nam na (prawie) każdym kroku, są nieodłącznym elementem współczesnego świata.

Zapamiętaj

Przedmiot AiPP łączy teorię (algorytmy) z praktyką (programowanie). Pamiętaj, że nie da się nabrać biegłości w żadnej dziedzinie życia bez intensywnych, systematycznych ćwiczeń.

Nie szczędź więc swego czasu na własne próby zmierzenia się z przedstawionymi problemami, eksperymentuj. Przecież to praktyka mistrza czyni.

W tym semestrze poznasz najbardziej podstawowe zagadnienia informatyki. Skrypt, którego celem jest systematyzacja Twej wiedzy, podzielony jest na 8 części:

1. Etapy tworzenia oprogramowania. Algorytm
2. Podstawy organizacji i działania komputerów. Reprezentacja liczb całkowitych i zmiennopozycyjnych
3. Deklaracja zmiennych w języku C++. Operatory arytmetyczne, logiczne i relacyjne.
4. Instrukcja warunkowa i pętle
5. Funkcje. Przekazywanie parametrów przez wartość i przez referencję. Rekurencja
6. Wskaźniki. Dynamiczna alokacja pamięci. Tablice jednowymiarowe. Sortowanie. Łańcuchy znaków
7. Macierze
8. Struktury w języku C++. Abstrakcyjne typy danych

W niniejszej części zastanowimy się, w jaki sposób — w wyidealizowanym przypadku — powstają programy komputerowe. Co ważne, niebawem i Tym będziesz postępował(a) mniej więcej w przedstawiony niżej sposób rozwiązując różne problemy z użyciem komputera.

1.2. Etapy tworzenia oprogramowania

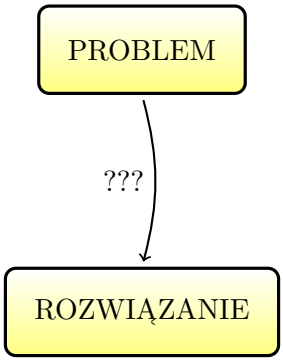
1.2.1. Sformułowanie i analiza problemu

1.2.1.1. Sformułowanie problemu

Punktem wyjścia naszych rozważań niech będzie pewne *zagadnienie problemowe*. Problem ten może być bardzo ogólnej natury, np. numerycznej, logicznej, społecznej czy nawet egzystencjalnej. Oto kilka przykładów:

- obliczenie wartości pewnego wyrażenia arytmetycznego,
- znalezienie rozwiązania układu 10000 równań różniczkowych celem znalezienia optymalnej trajektorii lotu pocisku, który uchroni ludzkość od zagłady,
- zaplanowanie najkrótszej trasy podróży od miasta X do miasta Y,
- postawienie diagnozy medycznej na podstawie listy objawów chorobowych,
- rozpoznanie twarzy przyjaciół na zdjęciu,
- dokonanie predykcji wartości indeksu giełdowego,
- zaliczenie (na piątkę, rzecz jasna) przedmiotu AiPP,
- rozwiązanie dylematu filozoficznego, np. czym jest szczęście i jak być szczęśliwym?

Znalezienie *rozwiązania* danego problemu jest dla nas często — z różnych względów — bardzo istotne. Kluczowym pytaniem jest tylko: jak to zrobić? Sytuację tę obrazuje rys. 1.1. Słowem: interesuje nas, w jaki sposób *dojść* do celu?



Rys. 1.1. Punkt wyjścia naszych rozważań

1.2.1.2. Analiza problemu

Po sformułowaniu problemu, czyli stwierdzeniu, że odczuwamy pilną potrzebę uzyskania odpowiedzi na postawione sobie pytanie, przechodzimy do etapu *analizy*. Tutaj doprecyzowujemy, o co nam naprawdę chodzi, co rozumiemy pod pewnymi pojęciami, jakich wyników się spodziewamy i do czego ewentualnie mogą w przyszłości się one nam przydać.

W przypadku pewnych zagadnień (np. matematycznych) zadanie czasem wydaje się względnie proste. Wszystkie pojęcia mają swoją definicję formalną, można udowodnić, że pewne kroki prowadzą do spodziewanych, jednoznacznych wyników itd.

Jednakże niektóre zagadnienia mogą przytłaczać swoją złożonością. Na przykład „zaliczenie przedmiotu AiPP” wymaga m.in. określenia:

- jaki stan końcowy jest pożądany (ocena bardzo dobra?),
- jakie czynności są kluczowe do osiągnięcia rozwiązania (udział w ćwiczeniach i laboratoriach, słuchanie wykładu, zadawanie pytań, dyskusje na konsultacjach),
- jakie czynniki mogą wpływać na powodzenie na poszczególnych etapach nauki (czytanie książek, wspólna nauka?), a jakie je wręcz uniemożliwiać (codzienne imprezy? brak prądu w akademiku? popsuty komputer?).

Komputery. Istotną cechą wielu zagadnień jest to, że nadają się do rozwiązania za pomocą *komputera*. Jak pokazuje rozwój współczesnej informatyki, tego typu problemów jest wcale niemało. W innych przypadkach często mamy do czynienia z sytuacją, w której komputery mogą pomóc uzyskać *rozwiązanie częściowe* lub zgrubne przybliżenie rozwiązania, które przecież może być lepsze niż zupełny brak rozwiązania.

Często słyszymy o niesamowitych „osiągnięciach" komputerów, np. wygraniu w szachy z mistrzem świata, uczestnictwie w pełnym podchwytliwych pytań teleturnieju *Jeopardy!* (pierwowzór niegdyś emitowanego w Polsce *Va Banque*), samodzielnym sterowaniu samochodem po zatłoczonych drogach (osiągnięcie ekipy *Google*), wirtualną obsługą petenta w Zakładzie Ubezpieczeń Społecznych itp. Wyobraźnię o ich potędze podsycają opowiadania *science-fiction*, których autorzy przepowiadają, że te maszyny pewnego dnia będą potrafiły zrobić prawie wszystko, o czym tylko jesteśmy w stanie pomyśleć.

Niestety, z drugiej strony komputery podlegają trzem bardzo istotnym ograniczeniom. By łatwiej można było je sobie uzmysłowić, posłużymy się analogią z dziedziny motoryzacji.

1	Komputer ma *ograniczoną moc obliczeniową*. Każda instrukcja wykonuje się przez pewien (niezerowy!) czas. Im bardziej złożone zadanie, tym jego rozwiązywanie trwa dłużej. Choć stan rzeczy poprawia się wraz z rozwojem technologii, zawsze będziemy ograniczeni zasadami fizyki.	Samochody mają np. ograniczoną prędkość, ograniczone przyspieszenie (także zasadami fizyki).
2	Komputer „rozumie" określony *język* (języki), do którego *syntaktyki* (składni) trzeba się dostosować, którego konstrukcję trzeba poznać, by móc się z nim „dogadać". Języki są zdefiniowane formalnie za pomocą ściśle określonej gramatyki. Nie toleruje on najczęściej żadnych odstępstw lub toleruje tylko nieliczne.	Aby zwiększyć liczbę obrotów silnika, należy wykonać jedną ściśle określoną czynność — wcisnąć mocniej pedał gazu. Nic nie da uśmiechnięcie się bądź próba sympatycznej konwersacji z deską rozdzielczą na temat zalet jazdy z inną prędkością.
3	Komputer ograniczony jest ponadto przez tzw. *czynnik ludzki* — mówi się, że jest tak mądry, jak jego programista. Potrafi zrobić tylko to (i aż tyle), co sami mu dokładnie powiemy, co ma zrobić, krok po kroku, instrukcja po instrukcji. Nie domyśli się, co tak naprawdę nam chodzi po głowie. Każdy wydawany rozkaz ma bowiem określoną *semantykę* (znaczenie). Komputer jedynie potrafi go posłusznie wykonać.	Samochód zawiezie nas, gdzie chcemy, jeśli będziemy odpowiednio posługiwać się kierownicą i innymi przyrządami. Nie będzie protestował, gdy skręcimy nie na tym skrzyżowaniu, co trzeba. Albo gdy wjedziemy na drzewo. Bądź dwa.

Zapamiętaj

Celem przewodnim naszych rozważań w tym semestrze jest więc takie poznanie natury i *języka* komputerów, by mogły zrobić *dokładnie* to, co *my* chcemy. Oczywiście skupimy się tutaj tylko (i aż) na dość prostych problemach.

Na początek, dla porządku, przedyskutujemy definicję obiektu naszych zainteresowań. Otóż słowo *komputer* pochodzi od łacińskiego czasownika *computare*, który oznacza *obliczać*. Jednak powiedzenie, że komputer zajmuje się tylko obliczaniem, to stanowczo za wąskie spojrzenie.

Informacja

Komputer to *programowalne* urządzenie *elektroniczne* służące do *przetwarzania informacji*.

Aż cztery słowa w tej definicji wymagają wyjaśnienia. *Programowalność* to omówiona powyżej zdolność do przyjmowania, interpretowania i wykonywania wydawanych poleceń zgodnie z zasadami syntaktyki i semantyki używanego języka.

Po drugie, pomimo licznych eksperymentów przeprowadzanych m.in. przez biologów molekularnych i fizyków kwantowych, współczesne komputery to w znakomitej większości maszyny *elektroniczne*. Ich „wnętrzności" są w swej naturze więc dość nieskomplikowane (ot, kilkaset milionów tranzystorów upakowanych na płytce drukowanej). O implikacjach tego faktu dowiemy się więcej z drugiego wykładu poświęconego organizacji i działaniu tych urządzeń.

Dalej, przez jednostkę *informacji* rozumiemy dowolny *ciąg liczb lub symboli* (być może jednoelementowy). Oto kilka przykładów:

- (6) (liczba naturalna),
- (*PRAWDA*) (wartość logiczna),
- („STUDENT MiNI") (napis, czyli ciąg znaków drukowanych),
- (3,14159) (liczba rzeczywista),
- (4,-3,-6, 34) (ciąg liczb całkowitych),
- (011100) (liczba w systemie binarnym),
- ("WARSZAWA", 52°13'56"N, 21°00'30"E) (współrzędne GPS pewnego miejsca na mapie).

Ciekawostka

Wartym zanotowania faktem jest to, iż wiele obiektów spotykanych na co dzień może mieć swoje *reprezentacje* w postaci ciągów liczb lub symboli. Często te reprezentacje nie muszą odzwierciedlać wszystkich cech opisywanego obiektu lecz tylko te, które są potrzebne w danym zagadnieniu. Ciąg („Maciek Pryk", „Matematyka II rok", 4,92, 21142) może być wystarczającą informacją dla pani z dziekanatu, by przyznać pewnemu studentowi stypendium za wyniki w nauce. W tym przypadku kolor oczu Maciusia czy imię dziewczyny, do której wzdycha on w nocy, to zbędne szczegóły (chyba, że ta ostatnia jest córką Pani z dziekanatu, ale...).

I wreszcie, *przetwarzanie* informacji to wykonywanie różnych działań na liczbach (np. operacji arytmetycznych, porównań) lub symbolach (np. zamiana elementów, łączenie ciągów). Rodzaje wykonywanych operacji są ściśle określone, co nie znaczy, że nie można próbować samodzielnie tworzyć nowych na podstawie tych, które są już dostępne.

Jako przykładowe operacje przetwarzające, odpowiednio, liczby naturalne, wartości logiczne i napisy, możemy przytoczyć:

$$\begin{array}{rcc} 2+2 & \to & 4 \\ \pi < e & \to & \mathit{FAŁSZ} \\ \text{"KATA"} \oplus (\text{"STREFA"}/(\text{E} \rightsquigarrow \text{O})) & \to & \text{"KATASTROFA"} \end{array}$$

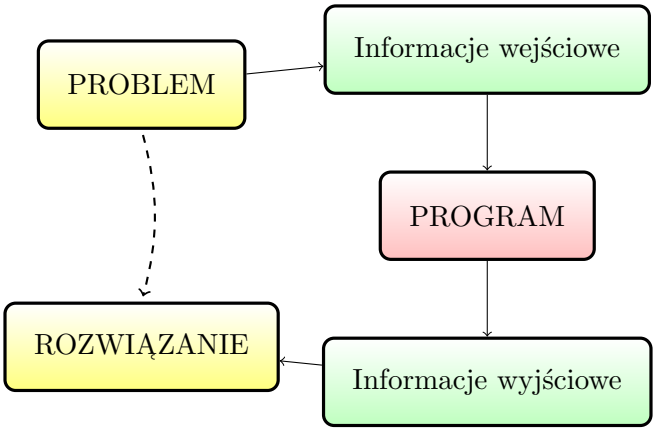
Nietrudno odgadnąć ich znaczenie. Co ważne: właśnie tak działa komputer!

Zapamiętaj

Ciąg operacji, które potrafi wykonać komputer, nazywamy *programem komputerowym*.

Dokonajmy syntezy naszych dotychczasowych rozważań. Otóż *problem, który da się rozwiązać za pomocą komputera* posiada dwie istotne cechy (por. rys. 1.2):

1. daje się wyrazić w postaci pewnych danych (informacji) wejściowych,
2. istnieje dla niego program komputerowy, przetwarzający dane wejściowe w taki sposób, że informacje wyjściowe mogą zostać przyjęte jako reprezentacja poszukiwanego rozwiązania.



Rys. 1.2. Rozwiązanie problemu za pomocą programu komputerowego

1.2.2. Projektowanie

Po etapie analizy problemu następuje etap *projektowania algorytmów*.

Zapamiętaj

Algorytm to abstrakcyjny przepis (proces, metoda, „instrukcja obsługi") pozwalający na uzyskanie, za pomocą *skończonej* liczby działań, oczekiwanych danych wyjściowych na podstawie poprawnych danych wejściowych.

Przykładowym algorytmem „z życia" jest przepis na pierniczki,[1] przedstawiony w tab. 1.1.

Tab. 1.1. Przepis na pierniczki

Dane wejściowe: 2 szklanki mąki; 2 łyżki miodu; 3/4 szklanki cukru; 1,5 łyżeczki sody oczyszczonej; 1/2 torebki przyprawy piernikowej; 1 łyżka masła; 1 jajko (+ dodatkowo 1 jajko do posmarowania); około 1/3 szklanki lekko ciepłego mleka.

Algorytm:

1. Mąkę przesiać na stolnicę, wlać rozpuszczony gorący miód i wymieszać (najlepiej nożem). Ciągle siekając, dodawać kolejno cukier, sodę, przyprawy, a gdy masa lekko przestygnie – masło i jedno jajko.
2. Dolewając stopniowo (po 1 łyżce) mleka zagniatać ręką ciasto aż będzie średnio twarde i gęste (nie musimy wykorzystać całego mleka, bo masa może być za rzadka). Dokładnie wyrabiać ręką, aż będzie gładkie, przez około 10 minut.
3. Na posypanej mąką stolnicy rozwałkować ciasto na placek o grubości maksymalnie 1 cm. Foremkami wykrajać z ciasta pierniczki, smarować rozmąconym jajkiem i układać na blasze wyłożonej papierem do pieczenia w odstępach około 2–3 cm od siebie (pierniczki troszkę podrosną).
4. Piec w piekarniku nagrzanym do 180 stopni przez około 15 minut. Przechowywać w szczelnie zamkniętym pojemniku, do 4 tygodni lub jeszcze dłużej [*oj tam! — przyp. MG*]. Pierniczki im są starsze tym lepsze. Z czasem też stają się bardziej miękkie.
5. Dekorować przed podaniem, najlepiej jak już będą miękkie. Do dekoracji można użyć samego lukru lub lukru wymieszanego z barwnikiem spożywczym. Zamiast barwnika spożywczego można użyć soku z granatu lub z buraka. Pierniczki można dekorować roztopioną czekoladą i maczać w posypce cukrowej lub w wiórkach kokosowych.

Dane wyjściowe: Pyszne pierniczki.

Widzimy, jak wygląda *zapisany* algorytm. Ważną umiejętnością, którą będziemy ćwiczyć, jest odpowiednie jego *przeczytanie*. Dobrze to opisał D. E. Knuth (2002, s. 4):

> *Na wstępie trzeba jasno powiedzieć, że algorytmy to nie beletrystyka. Nie należy czytać ich ciurkiem. Z algorytmem jest tak, że jak nie zobaczysz, to nie uwierzysz. Najlepszą metodą poznania algorytmu jest wypróbowanie go.*

A zatem — do kuchni!

[1] Przepis ten pochodzi z serwisu Kwestia Smaku:
www.kwestiasmaku.com/desery/ciasteczka/pierniczki/przepis.html

Informacja

Oto najistotniejsze *cechy algorytmu* (por. Knuth, 2002, s. 4):

- skończoność — wykonanie algorytmu musi zatrzymać się po skończonej liczbie kroków;
- dobre zdefiniowanie — każdy krok algorytmu musi być opisany precyzyjne, ściśle i jednoznacznie, tj. być sformułowanym na takim poziomie ogólności, by każdy, kto będzie go czytał, był w stanie zrozumieć, jak go wykonać;
- efektywność — w algorytmie nie ma operacji niepotrzebnie wydłużających czas wykonania;
- dane wejściowe są ściśle określone, pochodzą z dobrze określonych zbiorów;
- dane wyjściowe, czyli wartości powiązane z danymi wejściowymi, odpowiadają specyfikacji oczekiwanego poprawnego rozwiązania.

Wygląda na to, że nasz przepis na pierniczki posiada wszystkie wyżej wymienione cechy. Wykonując powyższe czynności ciasto to uda nam się kiedyś zjeść (skończoność). Wszystkie czynności są zrozumiałe nawet dla mało wprawionej gospodyni (aczkolwiek w *dobre zdefiniowanie* może tutaj trochę matematyk powątpiewać — co to znaczy *około* 1/3 szklanki ciepłego mleka?). Sam proces przygotowania jest efektywny (nie każe kucharce np. umalować się w trakcie mieszania składników bądź pojechać w międzyczasie na zagraniczną wycieczkę). Dane wejściowe i wyjściowe są dobrze określone.

Zapamiętaj

Jeden program rozwiązujący rozpatrywany problem może zawierać realizację *wielu algorytmów*, np. gdy złożoność zagadnienia wymaga podzielenia go na kilka *podproblemów*.

I tak zdolna kucharka wykonująca program „obiad” powinna podzielić swą pracę na podprogramy „rosół”, „chili con carne z plackami tortilli” oraz „pierniczki” i skupić się najpierw na projektowaniu podzadań.

Ważne jest, że algorytm nie musi być wyrażony za pomocą języka zrozumiałego dla komputera. Taki sposób opisu algorytmów nazywa się często *pseudokodem*. Stanowi on etap pośredni między analizą problemu a implementacją, opisaną w kolejnym paragrafie. Pseudokod ma po prostu pomóc w bardziej formalnym podejściu do tworzenia programu.

Dla przykładu, w przytoczonym wyżej przepisie, nie jest dokładnie wytłumaczone — krok po kroku — co oznacza „aż będzie średnio twarde i gęste”. Jednakże czynność tę da się pod pewnymi warunkami doprecyzować.

Jakby tego było mało, może istnieć wiele algorytmów służących do rozwiązania tego samego problemu! Przyjrzyjmy się następującemu przykładowi.

1.2.2.1. Przykład: Problem młodego Gaussa

Szeroko znana jest historia Karola Gaussa, z którym miał problemy jego nauczyciel matematyki. Aby zająć czymś młodego chłopca, profesor kazał mu wyznaczyć sumę liczb $a, a+1, \ldots, b$, gdzie $a, b \in \mathbb{N}$ (jak głosi historia, było to $a = 1$ i $b = 100$). Zapewne nauczyciel pomyślał sobie, że tamten użyje algorytmu I i spędzi niemałą ilość czasu na rozwiązywaniu łamigłówki.

Algorytm I wyznaczania sumy kolejnych liczb naturalnych

```
// Wejście: a,b ∈ ℕ (a < b)
// Wyjście: a + a + 1 + ... + b ∈ ℕ

niech suma, i ∈ ℕ;
suma = 0;

dla (i = a, a+1, ..., b)
    suma = suma + i;

zwróć suma jako wynik;
```

Jednakże sprytny Gauss zauważył, że $a + a + 1 + \cdots + b = \frac{a+b}{2}(b - a + 1)$. Dzięki temu wyznaczył wartość rozwiązania bardzo szybko korzystając z algorytmu II.

Algorytm II wyznaczania sumy kolejnych liczb naturalnych

```
// Wejście: a,b ∈ ℕ (a < b)
// Wyjście: a + a + 1 + ... + b ∈ ℕ

niech suma ∈ ℕ;
suma = (a+b)/2 (b − a + 1);
zwróć suma jako wynik;
```

Zauważmy, że *obydwa algorytmy rozwiązują poprawnie to samo zagadnienie*. Mimo to inna jest liczba operacji arytmetycznych $(+, -, *, /)$ potrzebna do uzyskania oczekiwanego wyniku. Poniższa tabelka zestawia tę miarę efektywności obydwu rozwiązań dla różnych danych wejściowych. Zauważmy, że w przypadku pierwszego algorytmu liczba wykonywanych operacji dodawania jest równa $(b - a + 1)$ [instrukcja $suma = suma + i$] $+ (b - a)$ [dodawanie występuje także w pętli `dla`...].

Tab. 1.2. Liczba operacji arytmetycznych potrzebna do znalezienia rozwiązania problemu młodego Gaussa.

a	b	Alg. I	Alg. II
1	10	19	5
1	100	199	5
1	1000	1999	5

Ciekawostka

Badaniem efektywności algorytmów zajmuje się dziedzina zwana *analizą algorytmów*. Czerpie ona obficie z wyników teoretycznych takich działów matematyki jak matematyka dyskretna czy rachunek prawdopodobieństwa. Z jej elementami zapoznamy się podczas innych wykładów.

1.2.3. Implementacja

Na kolejnym etapie abstrakcyjne algorytmy, zapisane często w postaci pseudokodu (np. za pomocą języka polskiego), należy przepisać w formie, która jest *zrozumiała* nie tylko przez nas, ale i *przez komputer*. Jest to tzw. *implementacja* algorytmów.

Intuicyjnie, tutaj wreszcie tłumaczymy komputerowi, co dokładnie chcemy, by zrobił, tzn. go *programujemy*. Efektem naszej pracy będzie *kod źródłowy* programu.

Ponadto, jeśli zachodzi taka potrzeba, na tym etapie należy dokonać scalenia czy też powiązania podzadań („rosół", „chili con carne" oraz „pierniczki") w jeden spójny projekt („obiad").

Zapamiętaj

Formalnie rzecz ujmując, zbiór zasad określających, jaki ciąg symboli tworzy kod źródłowy programu, nazywamy *językiem programowania*.

Reguły składniowe języka (ang. *syntactic rules*) ściśle określają, które (i tylko które) wyrażenia są poprawne. *Reguły znaczeniowe* (ang. *semantics*) określają precyzyjnie, jak komputer ma rozumieć dane wyrażenia. Dziedziną zajmującą się analizą języków programowania jest *lingwistyka matematyczna*.

W trakcie zajęć z AiPP będziemy poznawać *język C++*, zaprojektowany ok. 1983 r. przez B. Stroustrupa (aktualny standard: ISO/IEC 14882:2003). Należy pamiętać, że C++ jest tylko jednym z wielu (naprawdę wielu) sposobów, w jakie można wydawać polecenia komputerowi.

Ciekawostka

Język C++ powstał jako rozwinięcie języka C. Wśród innych, podobnych do niego pod względem składni, języków można wymienić takie popularne narzędzia jak Java, C#, PHP czy JavaScript.

Język C++ wyróżnia się zwięzłością składni i efektywnością generowanego kodu wynikowego. Jest ponadto jednym z najbardziej popularnych języków ogólnego zastosowania.

Kod źródłowy programu zapisujemy zwykle w postaci kilku plików tekstowych (tzw. plików źródłowych, ang. *source files*). Pliki te można edytować za pomocą dowolnego edytora tekstowego, np. *Notatnik* systemu Windows. Jednak do tego celu lepiej przydają się środowiska programistyczne. Na przykład, my podczas laboratoriów będziemy używać *Microsoft Visual C++*[2].

Zapamiętaj

Narzędziem, które pozwala przetworzyć pliki źródłowe (zrozumiałe dla człowieka i komputera) na kod maszynowy programu komputerowego (zrozumiały tylko dla komputera) nazywamy *kompilatorem* (ang. *compiler*).

Zanotujmy, że środowisko *Visual C++* posiada zintegrowany (wbudowany) kompilator tego języka.

[2]Dostępna jest bezpłatna wersja tego środowiska, zob. instrukcję instalacji opisaną w samouczku nr 1.

1.2.4. Testowanie

Gotowy program należy *przetestować*, to znaczy sprawdzić, czy dokładnie robi to, o co nam chodziło. Jest to, niestety, często najbardziej żmudny etap tworzenia oprogramowania. Jeśli coś nie działa, jak powinno, należy wrócić do któregoś z poprzednich etapów i naprawić błędy.

Warto zwrócić uwagę, że przyczyn niepoprawnego działania należy zawsze szukać w swojej omylności, a nie liczyć na to, że jakiś chochlik robi nam na złość. Jak powiedzieliśmy, komputer robi tylko to, co każemy. Zatem jeśli nakazaliśmy wykonać instrukcję, której skutków ubocznych nie jesteśmy do końca pewni, nasza to odpowiedzialność, by skutki te opanować.

1.2.5. Eksploatacja

Gdy program jest przetestowany, może służyć do rozwiązywania wyjściowego problemu (wyjściowych problemów). Czasem zdarza się, że na tym etapie dochodzimy do wniosku, iż czegoś nam brakuje lub że nie jest to do końca, o co nam na początku chodziło. Wtedy oczywiście pozostaje powrót do wcześniejszych etapów pracy.

Nauka programowania komputerów może nam pomóc rozwiązać wiele problemów. Problemów, których rozwiązanie w inny sposób wcale nie byłyby dla nas dostępne. A ponadto zobaczymy, że jest wspaniałą rozrywką!

1.2.6. Podsumowanie

Omówiliśmy następujące etapy tworzenia oprogramowania, które są niezbędne do rozwiązania zagadnień za pomocą komputera:

- sformułowanie i analiza problemu,
- projektowanie,
- implementacja,
- testowanie,
- eksploatacja.

Efekty pracy po każdym z etapów podsumowuje rys. 1.3.

Godne polecenia rozwinięcie materiału omawianego w tej części skryptu można znaleźć w książce Harela (2001, s. 9–31).

Na zakończenie przytoczmy fragment książki F.P. Brooksa (*Mityczny osobomiesiąc. Eseje o inżynierii oprogramowania*, wyd. WNT).

> Programowanie przynosi wiele radości; *Pierwsza to zwyczajna rozkosz tworzenia czegoś. Jak dziecko lubi stawiać domki z piasku, tak dorosły lubi budować coś, zwłaszcza według własnego projektu. [...] Druga to przyjemność robienia czegoś przydatnego dla innych ludzi. W głębi duszy chcemy, żeby inni korzystali z naszej pracy i uznawali ją za użyteczną. [...] Trzecia przyjemność to fascynacja tworzenia złożonych przedmiotów, przypominających łamigłówki składające się z zazębiających się, ruchomych części, i obserwowanie ich działania [...]. Czwarta przyjemność wypływa z ciągłego uczenia się i wiąże się z niepowtarzalnością istoty zadania. W taki czy inny sposób problem jest wciąż nowy, a ten, kto go rozwiązuje, czegoś się uczy [...]*

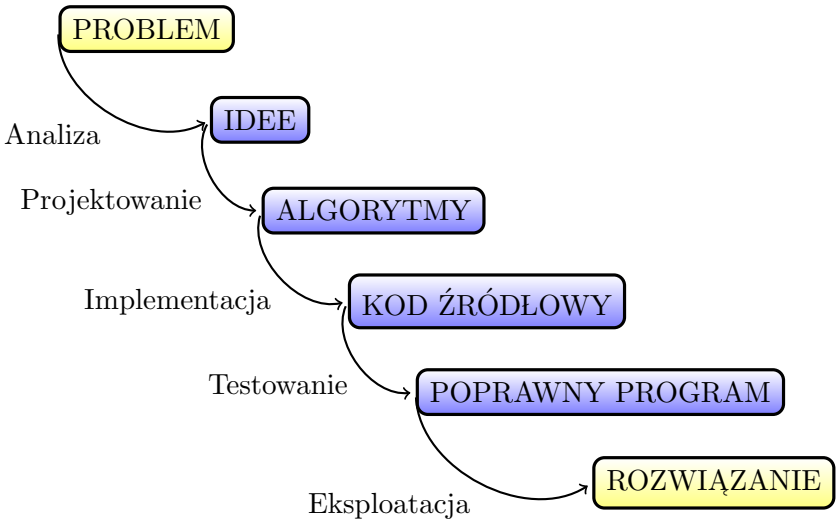
Brooks twierdzi także, że to, co przynosi radość, czasem musi powodować ból:

- należy działać perfekcyjnie,

– poprawianie i testowanie programu jest uciążliwe,

a także, w przypadku aplikacji komercyjnych:

– ktoś inny ustala cele, dostarcza wymagań; jesteśmy zależni od innych osób,
– należy mieć świadomość, że program, nad którym pracowało się tak długo, w chwili ukończenia najczęściej okazuje się już przestarzały.



Rys. 1.3. Efekty pracy po każdym z etapów tworzenia oprogramowania

1.3. Ćwiczenia

Zadanie 1.1. Rozważ poniższą implementację (w języku C++) algorytmu Euklidesa znajdywania największego wspólnego dzielnika (NWD) dwóch liczb naturalnych $0 \leqslant a < b$.

```
//  Wejście: 0 ⩽ a < b
//  Wyjście: NWD(a,b)
int NWD(int a, int b) // tj. NWD : ℕ × ℕ → ℕ
{
    assert(a >= 0 && a < b); // warunki poprawn. danych wej.
    int c;                   // niech c ∈ ℕ – zmienna pomocnicza
    while (a != 0)           // tzn. dopóki (a ≠ 0)
    {
        c = b % a;  // c staje się resztą z dzielenia b przez a
        b = a;
        a = c;
    }
    return b; // zwróć wartość b jako wynik;
}
```

Prześledź działanie algorytmu Euklidesa, prezentując w postaci tabelki, jakie wartości przyjmują zmienne a, b, c w każdym kroku. Rozpatrz: a) NWD(42, 56), b) NWD(192, 348), c) NWD(199, 544), d) NWD(2166, 6099).

Zadanie 1.2. Pokaż, w jaki sposób za pomocą ciągu operacji przypisania można przestawić wartości dwóch zmiennych (a, b) tak, by otrzymać (b, a).

Zadanie 1.3. Napisz kod, który przestawi wartości (a, b, c) na (c, a, b).

Zadanie 1.4. Napisz kod, który przestawi wartości (a, b, c, d) na (c, d, b, a).

Zadanie 1.5. Dokonaj permutacji ciągu (a, b, c) tak, by otrzymać ciąg (a', b', c') taki, że $a' \leqslant b' \leqslant c'$.

Zadanie 1.6. Pokaż, w jaki sposób za pomocą ciągu przypisań i podstawowych operacji arytmetycznych można mając na wejściu ciąg zmiennych (a, b, c) otrzymać a) $(b, c + 10, a/b)$, b) $(a+b, c^2, a+b+c)$, c) $(a-b, -a, c-a+b)$. Postaraj się zminimalizować liczbę użytych instrukcji i zmiennych pomocniczych.

Zadanie 1.7. [MD] Prześledź działanie poniższej funkcji w języku C++ wywołanej jako a) $f(2, 4)$ i b) $f(3, -1)$.

```
double f(double x, double y) // tj. f : ℝ × ℝ → ℝ
{
    double c;  // niech c ∈ ℝ – zmienna pomocnicza
    c = x + y + 1;
    if (y > 1) // jeśli y > 1
        c = c / y;
    return c;  // zwróć wartość c jako wynik
}
```

Zadanie 1.8. [MD] Wzorując się na kodzie z zadania 1.7 oraz wykorzystując funkcję f tam zdefiniowaną, napisz kod funkcji $g : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$, która jest określona wzorem:

$$g(a, b) = \begin{cases} f(a+b, a-b) & \text{dla } a > b, \\ b - a & \text{dla } b \geqslant a. \end{cases}$$

Zadanie 1.9. Dany jest ciąg n liczb rzeczywistych $\mathbf{x} = (x[0], \ldots, x[n-1])$ (umawiamy się, że elementy ciągów numerujemy od 0). Dla ciągów $(1, -1, 2, 0, -2)$ oraz $(34, 2, -3, 4, 3.5)$

prześledź działanie następującego algorytmu służącego do wyznaczania średniej arytmetycznej, która jest określona wzorem:

$$\frac{1}{n}\sum_{i=0}^{n-1} x[i] = \frac{1}{n}\left(x[0] + x[1] + \cdots + x[n-1]\right).$$

```
// Wejście: n > 0 oraz x[0], x[1], ..., x[n-1] ∈ ℝ
// Wyjście: średnia arytmetyczna z (x[0], x[1], ..., x[n-1])
niech s ∈ ℝ;
s = 0;
dla (i=0,1,...,n-1)
    s = s + x[i];
zwróc s/n jako wynik;
```

Zadanie 1.10. Dany jest ciąg n dodatnich liczb rzeczywistych $\mathbf{x} = (x[0], \ldots, x[n-1])$ (różnych od 0). Napisz algorytm, który wyznaczy ich średnią harmoniczną:

$$\frac{n}{\sum_{i=0}^{n-1} \frac{1}{x[i]}} = \frac{n}{1/x[0] + 1/x[1] + \cdots + 1/x[n-1]}.$$

Wyznacz za pomocą tego algorytmu wartość średniej harmonicznej dla ciągów $(1, 4, 2, 3, 1)$ oraz $(10, 2, 3, 4)$.

Zadanie 1.11. Napisz kod, który dla danych dodatnich liczb rzeczywistych a, b, c sprawdzi, czy może istnieć trójkąt prostokątny o bokach podanych długości.

Zadanie 1.12. Napisz kod, który dla danych dodatnich liczb rzeczywistych a, b, c sprawdzi, czy może istnieć trójkąt (dowolny) o bokach podanych długości.

Zadanie 1.13. Napisz kod, który dla danych liczb rzeczywistych a, b, c wyznaczy rozwiązanie równania $ax^2 + bx + c = 0$ (względem x).

⋆ **Zadanie 1.14.** [PS] „Szklana pułapka”. Masz do dyspozycji pojemniki na wodę o objętości x i y litrów oraz dowolną ilość wody w basenie. Czy przy ich pomocy (pojemniki wypełnione do pełna) można wybrać z litrów wody? Napisz pseudokod algorytmu, który to sprawdza i dokonaj obliczeń dla a) $x = 13, y = 31, z = 1111$, b) $x = 12, y = 21, z = 111$.

1.4. Wskazówki i odpowiedzi do ćwiczeń

Odpowiedź do zadania 1.1.
NWD(42,56) = 14.

```
5: a=42, b=56, c=0
7: a=42, b=56, c=14
8: a=42, b=42, c=14
9: a=14, b=42, c=14
5: a=14, b=42, c=14
7: a=14, b=42, c=0
8: a=14, b=14, c=0
9: a=0, b=14, c=0
5: a=0, b=14, c=0
Wynik: 14
```

NWD(192,348) = 12.

```
5: a=192, b=348, c=0
7: a=192, b=348, c=156
8: a=192, b=192, c=156
9: a=156, b=192, c=156
5: a=156, b=192, c=156
7: a=156, b=192, c=36
8: a=156, b=156, c=36
9: a=36, b=156, c=36
5: a=36, b=156, c=36
7: a=36, b=156, c=12
8: a=36, b=36, c=12
9: a=12, b=36, c=12
5: a=12, b=36, c=12
7: a=12, b=36, c=0
8: a=12, b=12, c=0
9: a=0, b=12, c=0
5: a=0, b=12, c=0
Wynik: 12
```

NWD(199, 544) = 1.
NWD(2166, 6099) = 57. □

Odpowiedź do zadania 1.9.
Wynik dla $(1, -1, 2, 0, -2)$: 0.
Wynik dla $(34, 2, -3, 4, 3.5)$: 8,1. □

Odpowiedź do zadania 1.10.
Wynik dla $(1, 4, 2, 3, 1)$: $\frac{60}{37}$.
Wynik dla $(10, 2, 3, 4)$: $\frac{240}{71}$. □

Odpowiedź do zadania ??. SKO$(5, 3, -1, 7, -2)$ = 59,2.

Podany algorytm wymaga $n^2 + 5n$ operacji arytmetycznych. Nie jest on efektywny, gdyż można go łatwo usprawnić tak, by potrzebnych było $5n + 1$ działań $(+, -, *, /)$. □