Programowanie witryn internetowych.

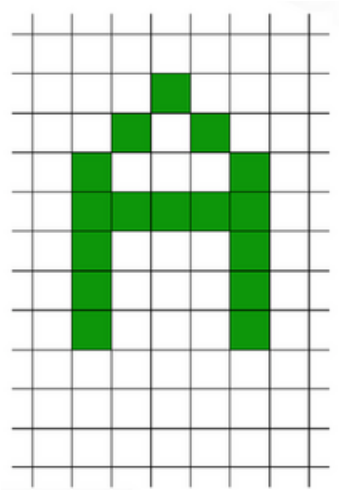
1. Grafika

Rodzaje grafiki komputerowej

Wyróżniamy dwa tryby generowania, reprezentowania oraz prezentowania obrazu:

Rastrowy

możliwe położenia piksela są z góry ustalone, sterować można jedynie jasnością (barwą) piksela



Wektorowy
możliwe jest dowolne sterowanie położeniem oraz jasnością piksela

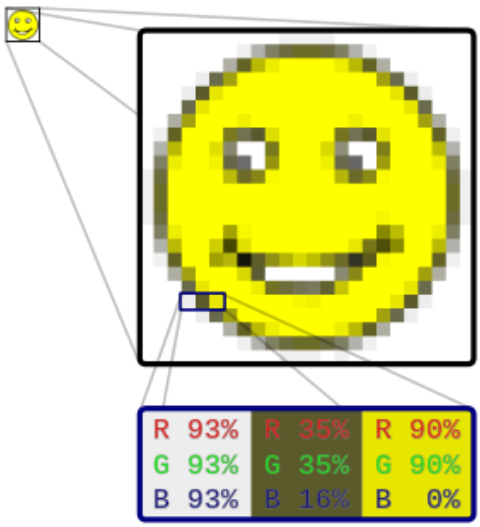
http://wazniak.mimuw.edu.pl/

•urządzenia wejścia/wyjścia

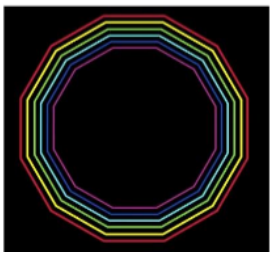
•Bitmapa, mapa bitowa – plik wykorzystujący rastrowy sposób polegający na określeniu położenia każdego piksela obrazu, oraz przypisaniu mu wartości bitowej określającej kolor w danym modelu barw

Rodzaje grafiki komputerowej

- brak możliwości skalowania (bez utraty jakości),
- odbiór rysunku zależny od urządzenia (linia widoczna w postaci schodków)
- możliwość operowania kreską i plamą.



- możliwość skalowania,
- odbiór rysunku niezależny od urządzenia (linia zawsze gładka)
- możliwość operowania kreską,
- trudności w operowaniu plamą.



2. Środowisko programisty.

Interpreter

Interpreter (program komputerowy)

Interpreter – program komputerowy, który analizuje kod źródłowy programu, a przeanalizowane fragmenty wykonuje.

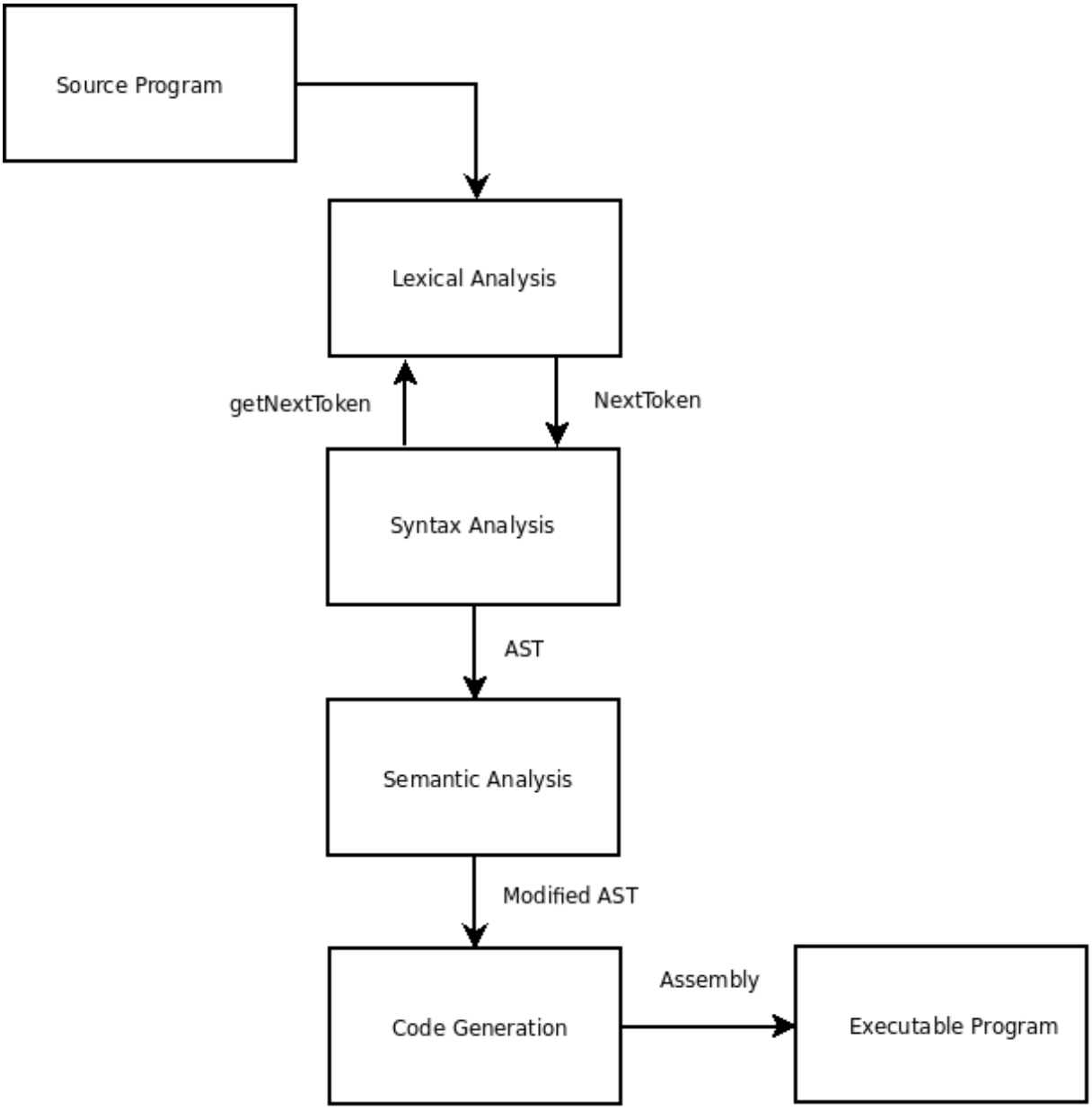
Realizowane jest to w inny sposób niż w procesie kompilacji, podczas którego nie wykonuje się wejściowego programu (kodu źródłowego), lecz tłumaczy go do wykonywalnego kodu maszynowego lub kodu pośredniego, który jest następnie zapisywany do pliku w celu późniejszego wykonania.

Wykonanie programu za pomocą interpretera jest wolniejsze, a do tego zajmuje więcej zasobów systemowych niż wykonanie kodu skompilowanego, lecz może zająć relatywnie mniej czasu niż kompilacja i uruchomienie. Jest to zwłaszcza ważne przy tworzeniu i testowaniu kodu, kiedy cykl edycja-interpretacja-debugowanie może często być znacznie krótszy niż cykl edycja-kompilacja-uruchomienie-debugowanie.

Interpretacja kodu programu jest wolniejsza od uruchamiania skompilowanego kodu, ponieważ interpreter musi wpierw przeanalizować każde wyrażenie i dopiero na tej podstawie wykonać odpowiednie akcje, a kod skompilowany wykonuje wyłącznie akcje. W implementacjach będących w pełni interpreterami wielokrotne wykonanie tego samego fragmentu kodu wymaga wielokrotnej interpretacji tego samego tekstu. Ta analiza nazywana jest "kosztem interpretacji". Dostęp do zmiennych jest także wolniejszy w przypadku interpretera, gdyż odwzorowanie identyfikatorów na miejsca w pamięci operacyjnej musi zostać dokonane podczas uruchomienia lub działania, a nie podczas kompilacji, dlatego niektóre interpretery tworzą dodatkowe dane (np. adresy zmiennych) przyspieszające wykonanie programu.

Kompilator

Kompilator – program służący do automatycznego tłumaczenia kodu napisanego w jednym języku (języku *źródłowym*) na równoważny kod w innym języku (języku *wynikowym*) [1]. Proces ten nazywany jest kompilacją. W informatyce kompilatorem nazywa się najczęściej program do tłumaczenia kodu źródłowego w języku programowania na język maszynowy. Niektóre z nich tłumaczą najpierw do języka asemblera, a ten na język maszynowy jest tłumaczony przez asembler.



Schemat blokowy kompilatora wieloprzebiegowego

Różnica pomiędzy kompilatorem a asemblerem polega na tym, iż każde polecenie języka programowania może zostać rozbite na wiele podpoleceń języka maszynowego (przy czym nowoczesne asemblery również posiadają składnię umożliwiającą zapis wielu poleceń maszynowych jako jednego polecenia kodu źródłowego oraz opcje optymalizacji kodu). Kompilatory mogą posiadać możliwość automatycznej alokacji pamięci dla zmiennych, implementowania struktur kontrolnych lub procedur wejścia-wyjścia.

Stosowanie kompilatorów ułatwia programowanie (programista nie musi znać języka maszynowego) i pozwala na większą przenośność kodu pomiędzy platformami.

Konwertery

Konwerterem kodu nazywamy program/układ służący do przekształcania jednego kodu w inny. Szczególnym przypadkiem konwertera jest dekoder przekształcający dowolny kod w kod "1 z N" oraz koder (zwany również enkoderem) zmieniający kod "1 z N" na żądany kod. Narzędzie wykorzystywane w językach maszynowych.

Język programowania – zbiór zasad określających, kiedy ciąg symboli tworzy program komputerowy oraz jakie obliczenia opisuje.

Konwerterem języka programowania nazywamy program służący do przekształcania jednego języka programowania w inny.

Interfejs użytkownika

Celem interfejsu użytkowego jest umożliwienie przekazywania do programu danych i odbierania od niego wyników oraz sterowania jego pracą.

Przedstawiony niżej przegląd obejmuje najważniejsze rodzaje interfejsów użytkowych. Począwszy od najprostszych, są to: standardowe strumienie danych wejściowych i wynikowych, czytanie i tworzenie plików, odczytywanie argumentów wywołania z wiersza poleceń, a na koniec — elementy sterowania aplikacją za pomocą wybranych elementów środowiska graficznego. Przegląd nie jest kompletny i nie wyczerpuje tematyki.

Interfejs strumieniowy

Dane wejściowe pobiera się za pomocą instrukcji `raw_input(`*`komunikat`*`)` (w Pythonie 3: `input(`*`komunikat`*`)`) albo w drodze czytania z pliku `sys.stdin`.

Wyniki wyprowadza się za pomocą instrukcji `print` *`wyrażenie`* (w Pythonie 3: `print(`*`wyrażenie`*`)`) lub poprzez zapisywanie ich do pliku `sys.stdout`.

Najważniejszą zaletą interfejsu strumieniowego jest łatwość przekierowania wejścia i wyjścia. Taki program może być używany w roli filtra.

Główną wadą interfejsu strumieniowego jest jego niska atrakcyjność dla użytkownika-niespecjalisty oraz fakt, że tego typu program operuje na jednym źródle danych oraz na jednym strumieniu wyników.

Kiedy program powinien być wyposażony w interfejs strumieniowy?

Przede wszystkim wtedy, kiedy chcemy, by nasz program był narzędziem współpracującym z innymi narzędziami typu filtr, np. `grep`, `sed`, `tr` itp.

Interfejs strumieniowy jest najprostszy do zaprojektowania, więc warto go stosować także w programach o jednym wejściu i jednym wyjściu, pisanych doraźnie dla jednokrotnego wykonania złożonej czynności.

Interfejs plikowy

Dane wejściowe są czytane z pliku o wskazanej nazwie.

Wyniki zapisywane są do pliku o wskazanej nazwie.

Nazwy plików mogą być określane przez użytkownika.

Kiedy program powinien być wyposażony w interfejs plikowy?

Przede wszystkim wtedy, kiedy chcemy, by nasz program czytał dane z plików, przetwarzał je i zapisywał dane wynikowe w plikach. Może to być program narzędziowy działający wsadowo, lub program interaktywny pobierający z pliku jedynie dane, zaś polecenia od operatora przyjmujący innym sposobem.

Wygodnym i często wykorzystywanym zastosowaniem interfejsu plikowego jest zarządzanie konfiguracją programu. Dane konfiguracyjne są odczytywane przy starcie programu z określonego pliku, zaś w chwili zmiany ustawień lub przy zakończeniu pracy plik ten jest nadpisywany.

Interfejs wiersza poleceń

Rozpoznawanie argumentów wiersza poleceń jest cenną właściwością wielu porządnie napisanych programów, nawet jeżeli ich podstawowy interfejs jest inny. Na przykład większość edytorów odczytuje argument wywołania jako nazwę pliku przeznaczonego do edycji.

Umożliwienie przekazywania programowi argumentów (np. nazw plików danych) za pomocą wiersza poleceń istotnie zwiększa elastyczność jego użytkowania. Pozwala to m.in. na wywoływanie go z innych programów z zamiarem przetworzenia wskazanego pliku, lub na przypisanie mu obsługi danego typu plików w środowisku graficznym. Program taki będzie też „intuicyjnie” reagował na upuszczenie na jego ikonkę ikonki pliku przeznaczonego do przetworzenia.

Elementy graficznego interfejsu użytkowego

Jednym z podstawowych celów graficznych środowisk użytkownika jest dostarczanie interfejsu dla programów użytkowych.

Program z interfejsem graficznym pobiera dane od użytkownika za pośrednictwem formularzy ekranowych, okien dialogowych, lub przez manipulację różnego typu obiektami graficznymi. W podobny sposób przedstawiane są dane wynikowe. Możliwa jest też prezentacja graficzna wyników.

Z punktu widzenia autora programu, obsługa elementów środowiska odbywa się przez użycie podprogramów i struktur danych zadeklarowanych w bibliotekach związanych z konkretnym środowiskiem. Są to na ogół duże i skomplikowane pakiety.

4. Języki programowania

Podobnie jak języki naturalne, język programowania składa się ze zbiorów reguł syntaktycznych oraz semantyki, które opisują, jak należy budować poprawne wyrażenia oraz jak komputer ma je rozumieć. Wiele języków programowania posiada pisemną specyfikację swojej składni oraz semantyki, lecz inne zdefiniowane są jedynie przez oficjalne implementacje.

Język programowania pozwala na precyzyjny zapis algorytmów oraz innych zadań, jakie komputer ma wykonać. W niektórych pracach pojęcie *języka programowania* jest ograniczane wyłącznie do tych języków, w których można zapisać wszystkie istniejące algorytmy – od strony matematycznej oznacza to, że język musi być przynajmniej kompletny w sensie Turinga[2], jednak można się także spotkać z wykorzystaniem tego pojęcia na określenie również bardziej ograniczonych języków.

Definicje

Język programowania może być zdefiniowany ze względu na kilka cech:

- *funkcja*: język programowania służy do tworzenia programów komputerowych, których zadaniem jest przetwarzanie danych, wykonywanie obliczeń i algorytmów oraz kontrolowanie/obsługa zewnętrznych urządzeń, np. drukarek, robotów itd.
- *przeznaczenie*: języki naturalne służą do komunikacji między ludźmi, natomiast języki programowania umożliwiają wydawanie poleceń maszynom. Niektóre z języków są wykorzystywane również do kontrolowania jednego urządzenia przez inne. Przykładowo, program wykonywany na komputerze może wygenerować kod PostScript do sterowania pracą drukarki bądź wyświetlacza.
- *konstrukcje składniowe*: język programowania może zawierać konstrukcje składniowe do manipulowania strukturami danych oraz zarządzania przepływem sterowania.
- *moc*: teoria obliczeń klasyfikuje języki według rodzajów obliczeń, które można za ich pomocą zrealizować (hierarchia Chomsky'ego). We wszystkich językach zupełnych w sensie Turinga da się zaimplementować ten sam zbiór algorytmów. Przykładem często stosowanego języka niezupełnego jest SQL służący do komunikacji z bazą danych.

Języki, w których nie da się realizować obliczeń (języki znaczników, jak HTML czy XML oraz gramatyki formalne, np. BNF), nie są zazwyczaj uznawane za języki programowania.

Przeznaczenie

Memetyczna ewolucja niektórych języków programowania według deklaracji autorów lub oficjalnych specyfikacji

Obecnie na świecie istnieją tysiące języków programowania i każdego roku powstają nowe. Od języków naturalnych odróżniają się wysoką precyzją oraz jednoznacznością. Człowiek podczas komunikacji między sobą stale popełnia niewielkie błędy lub pozostawia niedomówienia wiedząc, że drugi rozmówca najczęściej go zrozumie. Maszyny wykonują zadania dokładnie, dlatego każdą czynność trzeba opisać ściśle krok po kroku, ponieważ komputer nie potrafi dociec, co programista miał na myśli.

Wiele języków zostało zaprojektowanych od zera, lecz powszechna jest praktyka rozwijania już istniejących rozwiązań oraz celowego upodabniania jednego języka do innego. Pozwala to na szybsze opanowanie nowego języka przez programistów mających już doświadczenie w tworzeniu aplikacji. Potrzeba istnienia wielu różnorodnych języków wynika z dużej liczby sytuacji, w których są one wykorzystywane – każda posiada pewne specyficzne wymagania:

- wielkość programów waha się od niedużych skryptów pisanych przez amatorów do potężnych aplikacji rozwijanych przez setki programistów
- doświadczenie użytkowników waha się od nowicjuszy lub programistów okazjonalnych wymagających przede wszystkim prostoty, do ekspertów potrafiących zrobić użytek z oferowanych możliwości
- tworzone programy muszą spełniać określone wymagania dotyczące szybkości, skalowalności oraz wielkości
- istniejące języki mogą być zbyt rozbudowane do pewnych zadań
- programy mogą nie zmieniać się z biegiem lat lub być poddawane stałym modyfikacjom
- programiści mają różne gusta – każdy z nich ma swój ulubiony język, w którym pisze mu się najwygodniej

Z tych powodów nie powiodły się do dziś próby stworzenia języka uniwersalnego

Obecnie panuje tendencja do tworzenia języków umożliwiających rozwiązywanie problemów na wyższym poziomie abstrakcji. Pierwsze języki programowania były mocno związane z konkretnym sprzętem. Z biegiem czasu wynalezione zostały nowe techniki tworzenia oprogramowania znacznie poprawiające przenośność oraz opracowane algorytmy pozwalające automatycznie realizować zadania, którymi dotąd musiał zajmować się programista. Skraca to czas tworzenia aplikacji i zmniejsza liczbę okazji do popełnienia błędu, lecz w niektórych sytuacjach odbija się to negatywnie na wydajności (np. język Java).

Elementy języka

```
def add5(x):
   return x+5

def dotwrite(ast):
   nodename = getNodename()
   label=symbol.sym_name.get(int(ast[0]),ast[0])
   print '    %s [label="%s" % (nodename, label),
   if isinstance(ast[1], str):
      if ast[1].strip():
         print '= %s"];' % ast[1]
      else:
         print '"]'
   else:
      print '"];'
      children = []
      for  in n, childenumerate(ast[1:]):
         children.append(dotwrite(child))
      print ,'    %s -> {' % nodename
      for  in :namechildren
         print '%s' % name,
```

Kolorowanie składni jest często wykorzystywane w edytorach kodu do wizualnego różnicowania poszczególnych elementów składni, co ułatwia czytanie kodu. Na obrazku pokolorowany kod w języku Python.

Postać programu wyrażona w języku programowania określana jest jako kod źródłowy. Na język programowania składa się kilka elementów:

Składnia

Aby dany ciąg znaków mógł być rozpoznany jako program napisany w danym języku, musi spełniać pewne reguły, zwane składnią. Składnia opisuje:

- rodzaje dostępnych symboli
- zasady, według których symbole mogą być łączone w większe struktury

Składnia najczęściej opisywana jest w formalnym zapisie będącym połączeniem wyrażeń regularnych oraz notacji BNF lub EBNF. Poniżej przedstawiony jest przykład prostej gramatyki wzorowanej na języku Lisp:

```
wyrazenie ::= atom | lista
atom  ::= liczba | symbol
liczba  ::= [+-]?['0'-'9']+
symbol  ::= ['A'-'Za'-'z'].*
lista  ::= '(' wyrazenie* ')'
```

Zapis ten określa wygląd i budowę kolejnych symboli:

- wyrażeniem nazwiemy atom i listę
- atomem nazwiemy każdą liczbę lub symbol
- liczbą nazwiemy ciąg cyfr, który może zaczynać się opcjonalnie od znaku **+** lub **-**
- symbolem nazwiemy dowolną sekwencję dużych i małych liter alfabetu łacińskiego
- listą nazwiemy parę nawiasów, w której może się znaleźć zero lub więcej wyrażeń

Przykładowe ciągi, które spełniają podane reguły składni to: "*12345*", "*()*", "*(a b c232 (1))*".

Zauważmy, że na etapie przetwarzania składni w ogóle nie jest brane pod uwagę znaczenie poszczególnych symboli. W praktyce kod poprawny składniowo nie musi być poprawny semantycznie. Występuje tu analogia do języków naturalnych. Zdanie "Bździągwy się

muciosząˮ jest poprawne pod względem gramatycznym, lecz nie posiada żadnego znaczenia, ponieważ zostały w nim użyte nieistniejące słowa.

Semantyka

Semantyka języka programowania definiuje precyzyjnie znaczenie poszczególnych symboli oraz ich funkcję w programie. Semantykę najczęściej definiuje się słownie, ponieważ większość z jej zagadnień jest trudna lub wręcz niemożliwa do ujęcia w jakikolwiek formalizm. Część błędów semantycznych można wychwycić już w momencie wstępnego przetwarzania kodu programu, np. próbę odwołania się do nieistniejącej funkcji, lecz inne mogą ujawnić się dopiero w trakcie wykonywania.

Typy danych

Każdy język operuje na jakimś zestawie danych, dlatego niezbędne jest podzielenie danych na odpowiednie typy, zdefiniowane ich właściwości oraz operacji, jakie można na nich realizować. Większość języków posiada typy danych do reprezentowania:

- liczb całkowitych w różnych zakresach
- liczb zmiennoprzecinkowych (reprezentacje liczb rzeczywistych o różnym stopniu dokładności)
- ciągów tekstowych

Od strony sprzętowej wszystkie te informacje wyrażane są za pomocą sekwencji zer i jedynek. Język programowania nakłada jedynie odpowiednie ograniczenia i zasady ich przetwarzania. Zjawisko konwersji wartości jednego typu na inną nazywa się rzutowaniem.

Biblioteki standardowe

Dla większości języków zdefiniowana jest także biblioteka standardowa zawierająca podstawowy zestaw funkcji pozwalających realizować wszystkie najważniejsze operacje, np.:

- obsługę wejścia-wyjścia
- obsługę plików
- obsługę wielowątkowości
- zarządzanie pamięcią operacyjną
- podstawowe typy danych oraz funkcje do zarządzania nimi
- operacje na ciągach tekstowych

Użytkownicy traktują bibliotekę standardową często jako część języka, lecz od strony twórców są to osobne twory. Przykładowo, programiści piszący w języku D mają do dyspozycji zarówno oficjalną bibliotekę *Phobos*, jak i alternatywny projekt *Tango*.

Wykonywanie kodu

Aby program napisany w danym języku mógł być wykonany, niezbędne jest odpowiednie przetworzenie jego kodu źródłowego:

- kompilacja – kod źródłowy jest tłumaczony do postaci języka maszynowego, czyli sekwencji elementarnych operacji gotowych do bezpośredniego przetworzenia przez procesor komputera. Jeżeli dany język programowania podlega kompilacji, określany jest mianem *kompilowanego języka programowania*
- interpretacja – kod źródłowy jest na bieżąco tłumaczony i wykonywany przez dodatkowy program zwany interpreterem. Jeżeli język podlega interpretacji, nazywany jest *interpretowanym językiem programowania*

Wykonanie polegające na kompilacji do kodu maszynowego zapewnia najwyższą wydajność programom, lecz wygenerowany kod jest ściśle powiązany z platformą sprzętową. Ponadto kompilowane języki są bardziej zbliżone do sposobu funkcjonowania sprzętu, przez co programowanie w nich jest trudniejsze. Języki interpretowane zapewniają większą przenośność programów, które często są niezależne od platformy i systemu operacyjnego. Aby programy wyrażone w języku interpretowanym można było uruchomić na innej platformie, wystarczy napisać dla niej interpreter. Jednak taki sposób wykonywania odbija się negatywnie na wydajności. Alternatywnym rozwiązaniem jest kompilacja programów do postaci pośredniej, tzw. kodu bajtowego. Jest ona wykonywana przez wirtualne maszyny tłumaczące elementarne rozkazy kodu bajtowego na rozkazy procesora.

Klasyfikacja języków programowania

Języki programowania mogą być podzielone ze względu na:

- paradygmat programowania
- generację języka programowania
- sposób kontroli typów
- sposób wykonywania (kompilacja, interpretacja)
- poziom (języki niskiego poziomu są bardziej zbliżone pod względem budowy do działania sprzętu)
- przeznaczenie

Najpopularniejsze języki programowania

Lista dwudziestu najpopularniejszych języków programowania według TIOBE (stan na kwiecień 2015)[3]:

1. C
2. C++
3. Objective-C
4. Java
5. C#
6. (Visual) Basic
7. PHP
8. Python
9. JavaScript
10. Visual Basic .NET
11. Ruby
12. Transact-SQL
13. Perl
14. F#

15. Język asemblera
16. Lisp
17. PL/SQL
18. MATLAB
19. Delphi/Object Pascal
20. D

5. Składnia PHP

Oddzielanie instrukcji

Jak już można zauważyć w przykładach z poprzedniego rozdziału, jedną z głównych zasad języka PHP (jak i wielu innych – m. in. C i Perla) jest umieszczenie na końcu każdej instrukcji (niekoniecznie linii) znaku średnika (‚;’). Można go pominąć tylko jeśli w danym miejscu następuje przejście do trybu HTML, a więc po danej linii następuje symbol przejścia do trybu HTML.

Przykład 2.1. Oddzielanie instrukcji

```
<?php
    echo "To jest test";
?>

<?php echo "To jest test" ?>
```

Komentarze

Czasem zachodzi potrzeba oznaczenia czegoś w kodzie, dla kogoś innego czy nawet dla siebie samego (zwłaszcza jeśli pracuje się nad dużym projektem można się pogubić). Wtedy można skorzystać z jednej z kilku metod oznaczania, dzięki którym parser PHP będzie wiedział, że dany tekst nie jest częścią skryptu i można go zignorować. Komentarze przydają się także do tymczasowego „wyłączania” niektórych linii kodu. PHP obsługuje 3 metody oznaczania komentarzy – 2 z nich znane są z języków C/C++ a jedna z powłok (shell) systemów Uniksowych. Poniższe 2 metody służą do oznaczania, że tekst od danego miejsca do końca linii jest komentarzem:

Przykład 2.2. Stosowanie komentarzy

```
<?php

   echo "To jest test komentarzy"; // Ta metoda znana jest z języków C/C++

   echo "A to drugi test"; # A ta z powłok Uniksowych

?>
```

Ostatnia metoda, także znana z języków C/C++, służy do oznaczania wielu linii jako komentarz. Należy przy niej pamiętać, aby nie zagnieżdżać wewnątrz siebie takich komentarzy ponieważ może to doprowadzić do nieprawidłowego funkcjonowania skryptów.

Przykład 2.3. Komentarze wielolinijkowe

```
<?php
```

```
   echo "Test komentarzy"; /* Tu jest początek komentarza
   tu dalej trwa
   a tu się kończy */

?>
```

Zmienne

Jeśli ktoś nie spotkał się jeszcze z pojęciem zmiennej, to postaram się to wyjaśnić. Otóż zmienna jest to identyfikator znakowy, któremu przypisano jakąś wartość. W języku PHP zmienne oznacza się za pomocą znaku dolara (‚$') przed wspomnianym identyfikatorem. Obsługa zmiennych w PHP jest uproszczona do minimum. W „dużych" językach programowania zmienne trzeba najpierw inicjować (przy czym z góry trzeba określić typ zmiennej), zmienne tekstowe muszą mieć z góry ustalony rozmiar itp. W PHP nie jest to konieczne. Zmienna jest inicjalizowana (to znaczy rezerwowany jest dla niej pewien obszar w pamięci) przy pierwszym jej użyciu. Nazwy zmiennych muszą zaczynać się od litery (dużej lub małej) lub „underscore" (dolna kreska – ‚_') a dalej mogą się składać z dowolnej ilości liter, cyfr i znaków o kodzie ASCII powyżej 127. Przy nazwach zmiennych respektowana jest wielkość znaków – zmienne `$Test` i `$test` to dwie różne zmienne. Oto przykład przypisywania wartości zmiennym i wykorzystanie ich w poleceniu echo:

Przykład 2.4. Koncepcja zmiennych

```
<?php

   $nazwa = 1; // Zmiennej "nazwa" przypisywana jest wartość liczbowa 1

   $druga_nazwa = "Tekst"; // Zmiennej "druga_nazwa" przypisany jest ciąg
znaków "Tekst"

   $trzecia_nazwa = $nazwa; // Zmiennej "trzecia_nazwa" przypisywana
                            //jest wartość zmiennej "nazwa"

   echo "To jest $druga_nazwa"; // Powinien wyświetlić się napis "To jest
Tekst"

   echo '$druga_nazwa'; // Powinien wyświetlić się napis "$druga_nazwa"

   echo $nazwa; // Powinna wyświetlić się cyfra 1

?>
```

W powyższym przykładzie można zauważyć, że parametr dla polecenia echo można podawać zarówno w cudzysłowach jak i apostrofach. Jednak te parametry nie są sobie równoznaczne. W przypadku cudzysłowów zmienne zawarte między nimi są zamieniane na ich wartość, a w przypadku apostrofów zmienna pozostaje swoją nazwą (jak można zauważyć w powyższym przykładzie).

Typy zmienych

- liczby całkowite (integer)
- liczby rzeczywiste (double)
- ciągi (string)
- tablice (array)
- obiekty (object)

Dodatkowo PHP potrafi konwertować zmienne całkowite zapisane w różnych formatach liczbowych.

Przykład 2.5. Formaty liczbowe

```
<?php
$a = 1234; # liczba dziesiętna
$a = -123; # liczba ujemna
$a = 0123; # liczba ósemkowa (równoznaczne z dziesiętnym 83)
$a = 0x12; # liczba szesnastkowa (równoznaczne z dziesiętnym 18)
?>
```

Zmiana typu

Zazwyczaj nie jest konieczne określenie typu zmiennej – PHP sam to ustala, zależnie od kontekstu.

Przykład 2.6. Zmiana typu zmiennej

```
<?php
$blah = "0";  // $blah jest ciągiem (ASCII 48)
$blah++;      // $blah jest ciągiem "1" (ASCII 49)
$blah += 1;   // $blah jest teraz wartością całkowitą (2)
$blah = $foo + 1.3;  // $blah jest wartością rzeczywistą (1.3)
$blah = 5 + "10 Malutkich Świnek"; // $blah jest wartością całkowitą (15)
$blah = 5 + "10 Małych Świń";     // $blah jest wartością całkowitą (15)
?>
```

Podczas przypisywania zmiennej nowej wartości, poprzednia wartość jest oczywiście zamazywana. W takim przypadku typ zmiennej ustalany jest od nowa.

Jeśli jednak zachodzi potrzeba zmiany typu lub PHP błędnie rozpoznaje typ, to można tego dokonać za pomocą rzutowania (cast – efekt jest jednorazowy) lub za pomocą funkcji settype (efekt trwały).

Rzutowanie typów odbywa się przez podanie nowego typu w nawiasie przed zmienną lub wartością, której typ chcemy zmienić.

Przykład 2.7. Rzutowanie typów

```
<?php
$liczba_calkowita = 10;

$liczba_rzeczywista = (real) $liczba_calkowita;
?>
```

Dozwolone typy rzutowań

- (int), (integer) – rzutuj do typu całkowitego
- (real), (double), (float) – rzutuj do typu rzeczywistego
- (string) – rzutuj do ciągu
- (array) – rzutuj do tablicy
- (object) – rzutuj do obiektu

Drugim sposobem, trwałym, jest użycie funkcji `settype`. Funkcja ta pobiera 2 argumenty. Pierwszym jest nazwa zmiennej do ustalenia typu, a drugim ciąg określający nowy typ zmiennej.

Dopuszczalne argumenty funkcji settype

- „integer”
- „double”
- „string”
- „array”
- „object”

Funkcja zwraca wartość „true” gdy wszystko poszło pomyślnie. W przeciwnym razie zwracana jest wartość „false”.

Przykład 2.8. Przykład użycia funkcji settype

```
<?php

   $zmienna = 10.3;

   echo "$zmienna <br>"; // Wyświetlona wartość to "10.3"

   settype($zmienna, "integer");

   echo "$zmienna <br>"; // Wyświetlona wartość to "10"

?>
```

Przykład. Definicja funkcji

```
<?php

function suma($parametr1 = 0, $parametr2 = 0)
{

   $wartosc = $parametr1 + $parametr2;

   return $wartosc;

}

?>
```

Predefiniowane zmienne

W każdym skrypcie PHP dostępne jest kilka zmiennych, których wartość jest ustalana na podstawie zmiennych środowiskowych serwera WWW. Dostępne są jak zwykłe zmienne – ze znakiem dolara przed nazwą.

Zmienne ustawiane przez serwer WWW

`GATEWAY_INTERFACE`
: Informacja o specyfikacji CGI używanej przez serwer, np. ‚CGI/1.1’.

`SERVER_NAME`
: Nazwa hosta serwera na którym skrypt jest uruchamiany. Jeśli skrypt pracuje na wirtualnym hoście, to zmienna przyjmie jako wartość nazwę wirtualnego hosta.

`SERVER_SOFTWARE`
: Ciąg identyfikujący serwera podawany przy odpowiadaniu na zapytania.

`SERVER_PROTOCOL`
: Nazwa i numer wersji protokołu za pomocą którego wysłano zapytanie o stronę, np. ‚HTTP/1.0’;

`REQUEST_METHOD`
: Metoda zapytania użyta do uzyskania dostępu do strony, np. ‚GET’, ‚HEAD’, ‚POST’, ‚PUT’.

`QUERY_STRING`
: Ciąg zapytania (jeśli takowy istnieje) za pomocą którego połączono się ze stroną.

`DOCUMENT_ROOT`
: Katalog główny drzewa dokumentów spod którego skrypt jest wykonywany – jest to ustawienie z pliku konfiguracyjnego serwera.

`HTTP_ACCEPT`
: Nagłówek z aktualnego zapytania, jeśli taki istnieje.

`HTTP_ACCEPT_CHARSET`
: Zawartość nagłówka „Accept-Charset” z aktualnego zapytania, jeśli taki istnieje, np. ‚iso-8859-1,*,utf-8’.

`HTTP_ENCODING`

Zawartość nagłówka „Accept-Encoding” z aktualnego zapytania, jeśli taki istnieje, np. ‚gzip’.

`HTTP_ACCEPT_LANGUAGE`

Zawartość nagłówka „Accept-Language” z aktualnego zapytania, jeśli taki istnieje, np. ‚en’.

`HTTP_CONNECTION`

Zawartość nagłówka „Connection” z aktualnego zapytania, jeśli taki istnieje, np. ‚Keep-Alive’.

`HTTP_HOST`

Zawartość nagłówka „Host” z aktualnego zapytania, jeśli taki istnieje.

`HTTP_REFERER`

Adres strony (jeśli taka była), która wskazała przeglądarkę do tej strony. Wartość ta jest ustawiana przez przeglądarkę – nie wszystkie to robią.

`HTTP_USER_AGENT`

Zawartość nagłówka „User-Agent” z zapytania, jeśli taki istnieje. Jest to ciąg informujący o przeglądarce która została użyta do obejrzenia bieżącej strony, np. Mozilla/4.5 [en] (X11; U; Linux 2.2.9 i586). Można użyć funkcji get_browser() aby dopasować funkcjonalność strony do przeglądarki użytkownika.

`REMOTE_ADDR`

Adres IP z którego użytkownik połączył się z serwerem.

`REMOTE_PORT`

Port używany do komunikacji pomiędzy użytkownikiem a serwerem.

`SCRIPT_FILENAME`

Ścieżka do aktualnie wykonywanego skryptu.

`SERVER_ADMIN`

Wartość podana dla opcji SERVER_ADMIN w konfiguracji serwera WWW. Jeśli skrypt działa na wirtualnym serwerze, to będzie to wartość podana dla tego wirtualnego serwera.

`SERVER_PORT`

Port na serwerze którego użyto do połączenia. Dla normalnych połączeń będzie to ’80’.

`SERVER_SIGNATURE`

Ciąg zawierający wersję i nazwę wirtualnego hosta który jest dodawany do stron generowanych przez serwer.

`SCRIPT_NAME`

Zawiera ścieżkę do aktualnie wykonywanego pliku. Jest to przydatne do skyptów, które muszą wskazywać samego siebie.

`REQUEST_URI`

URI który został podany aby uzyskać dostęp do tej strony.

Zmienne ustawiane przez PHP

`argv`

Tablica argumentów przkazywanych do skryptu. Jeśli skrypt jest uruchamiany z linii poleceń, to zmienna ta daję dostęp do argumentów w stylu języka C. Jeśli jest wywołany przez metodę GET, to zmienna ta zawierać będzie ciąg parametrów (query string).

`argc`

Zawiera liczbę parametrów podanych podanych do skryptu w linii poleceń (jeśli skrypt został wywołany z linii poleceń).

`PHP_SELF`

Nazwa pliku aktualnie wykonywanego skryptu, względna do katalogu głównego dokumentów. Ta zmienna jest niedostępna jeśli PHP jest uruchamiany z linii poleceń.

`HTTP_COOKIE_VARS`
: Tablica asocjacjna zmiennych przekazanych do skryptu przez HTTP cookies. Dostępna tylko jeśli włączone zostało śledzenie zmiennych przez ustawienie w konfiguracji PHP opcji track_vars lub komendą `<?php_track_vars?>`.

`HTTP_GET_VARS`
: Tablica asocjacjna zmiennych przekazanych do skryptu przez metodę GET. Dostępna tylko jeśli włączone zostało śledzenie zmiennych przez ustawienie w konfiguracji PHP opcji track_vars lub komendą `<?php_track_vars?>`.

`HTTP_POST_VARS`
: Tablica asocjacjna zmiennych przekazanych do skryptu przez metodę POST. Dostępna tylko jeśli włączone zostało śledzenie zmiennych przez ustawienie w konfiguracji PHP opcji track_vars lub komendą `<?php_track_vars?>`.

Stałe

W PHP występują także tzw. stałe, czyli identyfikatory znakowe, których wartości nie można zmienić. Stałych, w odróżnieniu od zmiennych, używa się bez znaku dolara na początku. W PHP występuje kilka zmiennych ustawianych przez parser.

Stałe ustawiane przez PHP

`__FILE__`
: Nazwa pliku ze skryptem który jest aktualnie przetwarzany. Jeśli stała ta użyta jest wewnątrz pliku który został zainkludowany (o poleceniu include w dalszej części kursu), to podana zostanie nazwa pliku zainkludowanego, a nie pliku nadrzędnego.

`__LINE__`
: Numer linii w skrypcie która aktualnie jest przetwarzana. Jeśli stała ta użyta jest wewnątrz pliku który został zainkludowany, to podany zostanie numer linii przetwarzanej w pliku zainkludowanym.

`PHP_VERSION`
: Ciąg reprezentujący wersję parsera PHP aktualnie używaną.

`PHP_OS`
: Nazwa systemu operacyjnego na którym uruchamiany jest parser PHP.

`TRUE`
: Logiczna wartość prawdy.

`FALSE`
: Logiczna wartość fałszu.

Stałe mogą być definiowane przez użytkownika za pomocą funkcji `define()`, która przyjmuje 2 parametry: nazwę stałej i wartość do niej przypisaną.

Przykład Definiowanie stałych tekstowuch

```
<?php
  define("STALA", "Hello world.");
  echo STALA; // Wyświetla "Hello world."
?>
```

Przykład Definiowanie zmiennych ogólnie

```
<?php

   $nazwa = 1; // Zmiennej "nazwa" przypisywana jest wartość liczbowa 1

   $druga_nazwa = "Tekst"; // Zmiennej "druga_nazwa" przypisany jest ciąg
znaków "Tekst"

   $trzecia_nazwa = $nazwa; // Zmiennej "trzecia_nazwa" przypisywana
                            //jest wartość zmiennej "nazwa"

   echo "To jest $druga_nazwa"; // Powinien wyświetlić się napis "To jest
Tekst"

   echo '$druga_nazwa'; // Powinien wyświetlić się napis "$druga_nazwa"

   echo $nazwa; // Powinna wyświetlić się cyfra 1

?>
```

Zmiana i rzutowanie typów danych

Podczas przypisywania zmiennej nowej wartości, poprzednia wartość jest oczywiście zamazywana. W takim przypadku typ zmiennej ustalany jest od nowa.

Jeśli jednak zachodzi potrzeba zmiany typu lub PHP błędnie rozpoznaje typ, to można tego dokonać za pomocą rzutowania (cast – efekt jest jednorazowy) lub za pomocą funkcji settype (efekt trwały).

Rzutowanie typów odbywa się przez podanie nowego typu w nawiasie przed zmienną lub wartością, której typ chcemy zmienić.

Przykład . Rzutowanie typów

```
<?php
$liczba_calkowita = 10;

$liczba_rzeczywista = (real) $liczba_calkowita;
?>
```

Dozwolone typy rzutowań

- (int), (integer) – rzutuj do typu całkowitego
- (real), (double), (float) – rzutuj do typu rzeczywistego
- (string) – rzutuj do ciągu

- (array) – rzutuj do tablicy
- (object) – rzutuj do obiektu

Drugim sposobem, trwałym, jest użycie funkcji `settype`. Funkcja ta pobiera 2 argumenty. Pierwszym jest nazwa zmiennej do ustalenia typu, a drugim ciąg określający nowy typ zmiennej.

6. Procesy optymalizacji oprogramowania.

Debugowanie (z ang. *debugging*) – proces systematycznego redukowania liczby błędów w oprogramowaniu bądź systemie mikroprocesorowym, który zazwyczaj polega na kontrolowanym wykonaniu programu pod nadzorem debuggera.

Etymologia

Popularyzację słowa *bug* (z ang. robak, insekt), rozumianego jako błąd, przypisuje się zazwyczaj admirał Grace Hopper. Podczas prac nad komputerem Mark II na Uniwersytecie Harvarda jej współpracownicy znaleźli ćmę, która zaplątała się w przekaźnik, utrudniając działanie urządzenia. Admirał Hopper nazwała usunięcie martwego owada debugowaniem, czyli odrobaczeniem. Pojęciem tym posłużył się jednak już w roku 1878 Thomas Edison, który w jednym ze swoich listów określił słowem *bugs* usterki techniczne.

Proces debugowania

Chociaż każdy błąd wymaga indywidualnego podejścia, debugowanie można zazwyczaj podzielić na kilka etapów:

1. Reprodukcja błędu
2. Wyizolowanie źródła błędu
3. Identyfikacja przyczyny awarii
4. Usunięcie defektu
5. Weryfikacja powodzenia naprawy

Reprodukcja błędu

Odkrycia błędu może dokonać zarówno programista podczas rutynowych testów tworzonej funkcjonalności jak również tester bądź użytkownik. Po zgłoszeniu, błąd powinien zostać odtworzony na maszynie programisty, aby można było potwierdzić istnienie usterki. Zdarza się bowiem, że użytkownicy zgłaszają błędy dotyczące zamierzonego działania programu[1] bądź podają informacje niewystarczające do zaobserwowania defektu, jak np. niekompletny scenariusz interakcji, niedokładne dane wejściowe lub niepełne środowisko wykonania. Niemożność odtworzenia problemu znacznie utrudnia programiście dotarcie do jego przyczyny. Odnalezienie źródła błędu może w takiej sytuacji wymagać nadzorowania wykonania zdalnego procesu bądź analizy informacji pośrednich (np. zrzutu pamięci lub śladu wykonania). Brak możliwości reprodukcji problemu utrudnia również weryfikację powodzenia naprawy oraz testowanie nawrotów błędu w kolejnych wersjach programu.

Wyizolowanie źródła błędu

Kolejnym etapem debugowania jest eliminacja wszystkich tych czynników, które nie przyczyniają się bezpośrednio do powstania błędu. Dotyczy to zarówno zbędnych kroków scenariusza interakcji, jak i ilości danych wejściowych, których nadmiar może utrudnić dotarcie do źródła problemu. Eliminacja niepotrzebnych czynników ułatwia śledzenie duplikatów i jest tak bardzo istotna dla postępu pracy w dojrzałych systemach, że niektóre

zespoły nakłaniają testerów do upraszczania już znalezionych błędów zamiast zgłaszania nowych[2].

W przypadku istnienia metody samoczynnego wykonania programu i określenia wyniku jego uruchomienia, krok ten można w dużej mierze zautomatyzować. Dokonuje się tego przy pomocy wyszukiwania binarnego, ograniczając ilość danych tak długo, aż odjęcie żadnego z najmniejszych elementów wejścia nie spowoduje błędu wykonania[3].

Identyfikacja przyczyny awarii

Odnalezienie przyczyny awarii odbywa się zazwyczaj poprzez obserwację stanu programu podczas jego kontrolowanego uruchomienia. Do śledzenia wykorzystuje się zwykle specjalnie przygotowaną wersję programu. W przeciwieństwie do wersji udostępnianej klientom, debugowany program nie jest zaciemniony, sprawdza asercje oraz loguje najważniejsze zdarzenia i działania. Wykonanie programu można nadzorować przy pomocy debuggera, który umożliwia wstrzymywanie wykonania procesu oraz obserwację i modyfikację jego stanu. Ponadto, środowisko uruchomienia można wzbogacić o biblioteki, które dokonują ściślejszej kontroli dostępu do pamięci oraz śledzą jej alokację, aby wychwycić problemy, zanim pociągną za sobą kolejne.

Usunięcie defektu

Ustalenie źródła błędu nie musi kończyć się usunięciem jego objawów lub przyczyn. Na przykład, komisja kontroli technicznej (ang. *Technical Review Committee*) firmy Sun Microsystems przyznała, że brak metody `clone()` w interfejsie `Cloneable` jest niedopatrzeniem, ale zdecydowała nie zmieniać wadliwego typu, aby uniknąć problemów z kompilacją istniejących programów (pomimo ich niepoprawności)[4].

W systemach, nad którymi pracują duże zespoły, niezwykle istotne jest, aby zdawać sobie sprawę ze wszystkich konsekwencji, jakie wprowadzona zmiana może mieć dla pozostałych części systemu. Badania nad rozwojem oprogramowania dla central telefonicznych 5ESS, prowadzone przez naukowców z Bell Labs, wykazały, że poprawianie usterek znacznie częściej niż inne rodzaje aktywności wprowadza nowe błędy[5].

Ponieważ wadliwy fragment systemu może na skutek duplikacji występować w innych miejscach kodu źródłowego, należy również upewnić się, że korekta została zaaplikowana do wszystkich jego kopii.

Weryfikacja powodzenia naprawy

Proces debugowania kończy się sprawdzeniem, czy oryginalny scenariusz interakcji nie powoduje błędnego zachowania systemu. Ponadto, wymagane może być dokładniejsze zbadanie systemu w celu upewnienia się, czy naprawa nie wprowadziła innych, niechcianych efektów ubocznych. Następnie wykonuje się ewentualne testy regresji celem wykluczenia możliwości przywrócenia starszych błędów.

Uruchomienie pełnego zestawu testów dużego systemu może zabrać wiele czasu. W przypadku błędów związanych z bezpieczeństwem niektóre firmy decydują się na

udostępnienie nie w pełni przetestowanej nowej wersji swojego systemu, aby jak najszybciej zapobiec ewentualnym skutkom wykorzystania danej luki przez osoby niepowołane.

Narzędzia

Centralnym punktem procesu debugowania programu jest obserwacja jego wykonania w celu lokalizacji źródła usterki. Zadanie to ułatwiają narzędzia do dynamicznej analizy programu.

Debugger

Debugger umożliwia:

- wykonywanie programu w trybie pracy krokowej lub z zastawianiem tzw. pułapek (ang. breakpoints);
- podglądanie i ewentualną zmianę zawartości rejestrów, pamięci itd.

Osobny artykuł: Debugger.

Inne

Narzędzia śledzące alokację pamięci (np. *Valgrind*) ułatwiają odnajdywanie fragmentów programu odwołujących się do zwolnionych i niezaalokowanych obszarów pamięci jak również umożliwiają lokalizowanie wycieków pamięci.

2. Zmienna w programie.

Zmienna - konstrukcja programistyczna posiadająca trzy podstawowe atrybuty: symboliczną nazwę, miejsce przechowywania i wartość; pozwalająca w kodzie źródłowym odwoływać się przy pomocy nazwy do wartości lub miejsca 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[1]. 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.

Programowanie imperatywne polega w dużej mierze na modyfikowaniu wartości zmiennych na podstawie ich wcześniejszych wartości.

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 klasy (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.

W większości współczesnych języków zasięg jest statyczny (leksykalny) oznacza to podprogram ma dostęp do zmiennych lokalnych bloków w których jest zdeklarowany.

Przykład . Zmienna liczbowa całkowita (int)

```
int f () {
    int a;
    int g () {
        print(a);
    }
    ...
}
```

Funkcja g jest zadeklarowana wewnątrz funkcji f, w związku z tym ma dostęp do zmiennej lokalnej funkcji f - a. W niektórych językach (np. pierwsze implementacje LISPu) zasięg był dynamiczny, czyli nie było ważne gdzie funkcja jest zadeklarowana, tylko jaka funkcja ją wywołała. W poniższym przykładzie funkcja g wydrukuje zawartość zmiennej a z funkcji f.

Przykład . Zmienna funkcyjna

```
int g () {
    print(a);
}
int f() {
    int a;
    g();
}
```

Przykład. Inne definicje zakresu zmiennej

```
int - liczby całkowite
float - liczby zmiennoprzecinkowe pojedynczej precyzji
double - liczby zmiennoprzecinkowe podwójnej precyzji
```

Inne rodzaje zmiennych

- zmienna sterująca
- zmienna wbudowana
- zmienna nakładana

7. Procesy projektowania witryn internetowych (web design)

Web design – czynność polegająca na rozplanowaniu, zaprojektowaniu i wdrożeniu stron internetowych. Wymaga przemyślenia także takich elementów strony jak nawigacja, interaktywność, użyteczność, architektura informacji oraz współdziałanie elementów audio, tekstu, obrazków, hiperlinków oraz filmów[1].

Połączenie dobrego projektu z dobrze opracowaną hierarchią treści, zwiększa skuteczność strony jako kanału komunikacyjnego i źródła wymiany danych. Podnosi to możliwości kontaktu pomiędzy dostawcą i odbiorcą treści.

Metody web designu

Strony internetowe mogą powstawać:

- przez tworzenie plików tekstowych w HTML, PHP, ASP, Aspx, JavaScripcie, JSP, Pythonie, Ruby.
- przez użycie programu WYSIWYG do tworzenia stron (np. Dreamweaver).

Etapy realizacji [2]

- określenie wymagań dotyczących funkcjonalności
- określenie wymagań dotyczących formy graficznej
- określenie wymagań dotyczących elementów animacji
- wyznaczenie ram czasowych w oparciu o analizę wymagań
- wycena projektu, potwierdzenie specyfikacji przedstawionej klientowi
- akceptacja specyfikacji oraz wyceny przez klienta
- przygotowanie wstępnej wersji graficznej projektu w oparciu o wytyczne/materiały klienta
- naniesienie poprawek zgłoszonych przez klienta
- przygotowanie ostatecznej wersji graficznej projektu
- akceptacja ostatecznej wersji graficznej projektu
- rozpoczęcie pracy nad stroną programistyczną projektu.
- przedstawienie wersji testowej aplikacji.
- naniesienie ewentualnych poprawek dotyczących funkcjonalności projektu.
- integracja z zewnętrznymi systemami.
- akceptacja ostatecznej formy projektu przez klienta
- instalacja gotowego projektu na serwerze wskazanym przez klienta
- finalizacja projektu

Krok 1 - Definiowanie wymagań nowej witryny

Zespół projektowy, wraz z właścicielem witryny, precyzuje wymagania wobec nowej strony internetowej. Oczekiwania właściciela powinny uwzględniać oczekiwania potencjonalnego

użytkownika serwisu. Trzeba spojrzeć na tworzony projekt z punktu widzenia osoby, która będzie odwiedzać witrynę i spróbować odpowiedzieć na pytania:

- dlaczego taka osoba miałaby odwiedzić naszą stronę internetową,
- jakie znajdzie na niej informacje,
- co będzie mogła z nimi zrobić ?

Na tym etapie powinniśmy wiedzieć do kogo będzie kierowana strona internetowa, tak by mieć przeciętnego użytkownika witryny przed oczami, podczas kolejnych etapów tworzenia.

Należy także wyspecyfikować wszystkie techniczne rozwiązania potrzebne do osiągnięcia celu. Np. przewidywaną ilość osób odwiedzających witrynę, tak by w zależności od tej liczby przewidzieć odpowiednie rozwiązania serwerowe i wydajnościowe.

Krok 2 - Definiowanie zawartości witryny

Na tym etapie trzeba ustalić jakiego rodzaju informacje znają się na stronie i jak dużo ich będzie. Należy również zdefiniować, które obszary serwisu będą edytowalne, a w których treści będą umieszczone na stałe.

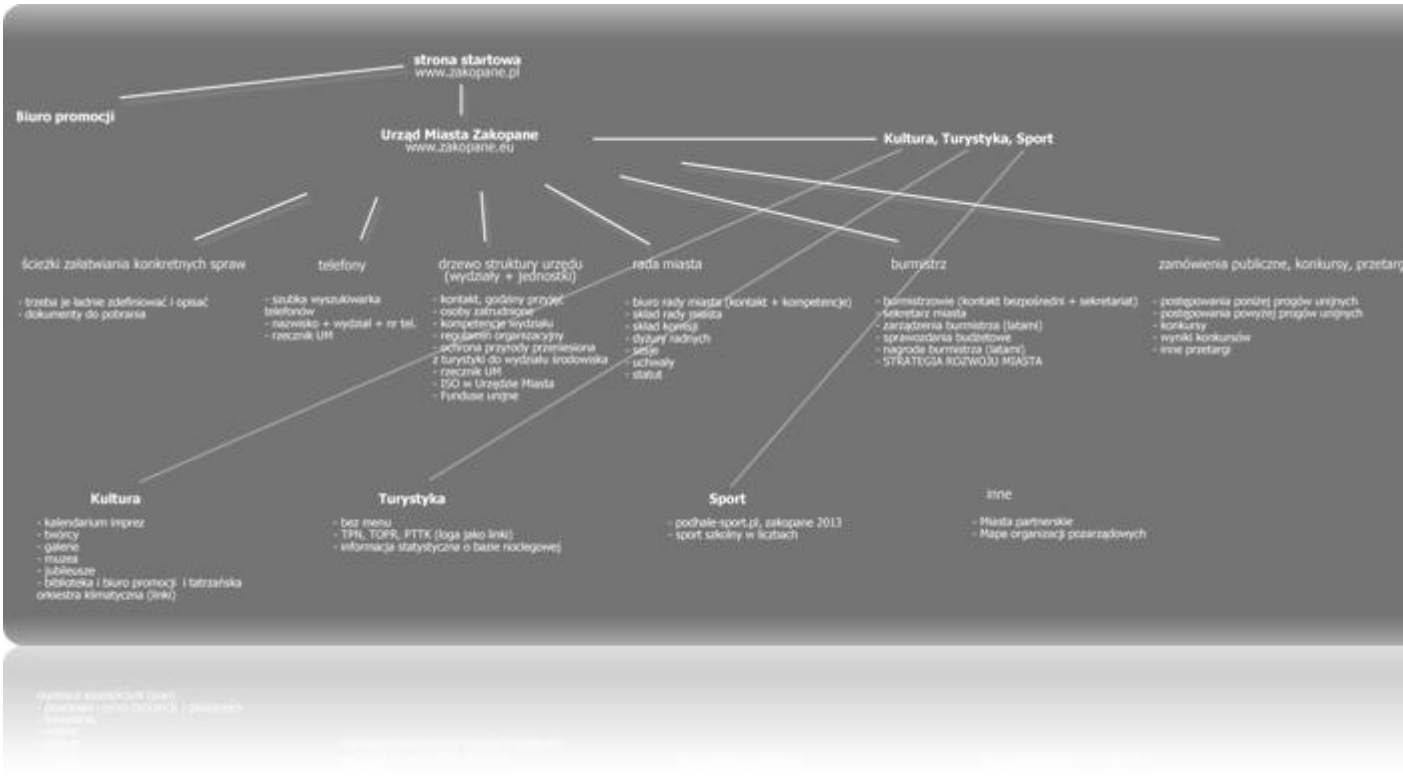
- teksty
- materiały graficzne
- fotografie
- elementy do pobrania (.pdf, .doc, .xls, .ppt ...)
- pliki video (.mov, .flv, .wmv, .avi, .ogg, youtube)

Powyższe materiały należy posegregować w odpowiednie kategorie, a plikom należy nadać nazwy, które będą korespondowały z ich zawartością. Jakość i logiczny podział meteriałów w znaczący sposób wpływa na czas wdrożenia serwisu. Jeśli materiały, które będą miały sie znaleźć na stronie internetowej zostaną dobrze posegregowane, to w kolejnym etapie (krok 3) dużo łatwiej można będzie wyznaczyć działy logiczne serwisu.

Rodzaj materiałów ma także wpływ na ruch generowany przez serwis. Jeśli na stronie będą umieszczane duże pliki graficzne czy też filmy, trzeba będzie przewidzieć dla takiej strony inne rozwiązanie hostingowe, niż dla strony gdzie będziemy umieszczać małe teksty z kilkoma zdjęciami.

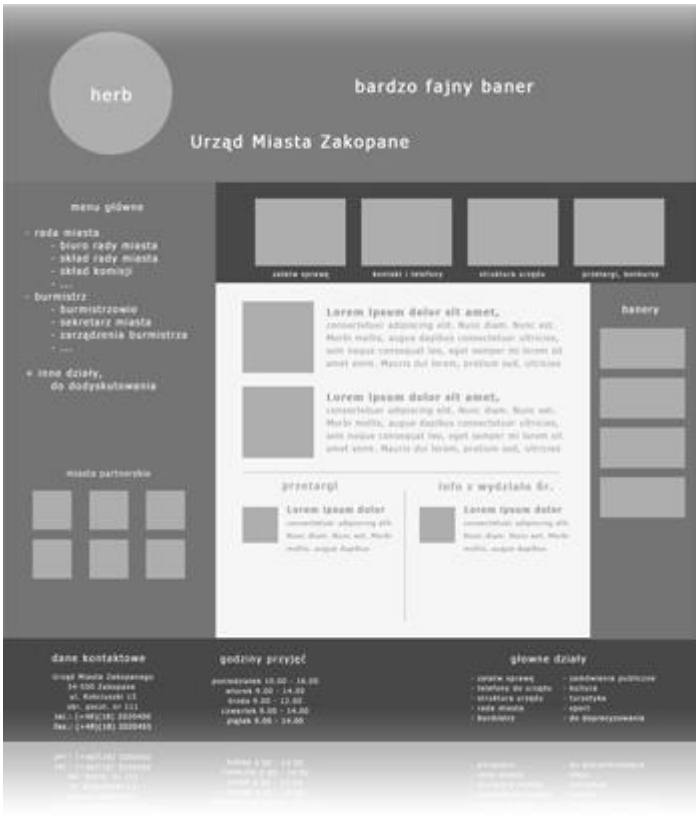
Krok 3 - Projektowanie struktury logicznej treści

Struktura logiczna treści to schemat, który pomaga nam ułożyć informacje tak, by były one czytelne dla potencjalnego użytkownika witryny. Właściciel witryny powinien zdecydować o priorytetach działów serwisu. Jeśli najważniejszy z całej strony ma być katalog produktów, czy dział z przetargami, musi to zostać zaznaczone.



Projekt struktury logicznej treści przedstawia również ścieżki, którymi będzie mógł podążać użytkownik witryny. Projekt przedstawia tylko główne ścieżki, ale nie wyklucza oczywiście możliwości przemieszczania się miedzy treściami strony poprzez linki w treściach serwisu, a także docierania do różnych treści poprzez wyszukiwarkę na stronie.

Krok 4 - Projektowanie układu logicznego witryny



Projekt układu logicznego witryny jest rozmieszczeniem elementów w konkretnych miejscach witryny, przeniesieniem struktury treści (krok 3) na konkretne elementy strony internetowej.

W zależności od zdefiniowanych w poprzednim kroku priorytetów, **wagi poszczególnych elementów** muszą one w układzie logicznym witryny zajmować odpowiednią przestrzeń.

Trzeba znaleźć odpowiednie miejsce na znaki identyfikacyjen instytucji czy organizacji (**logotypy**), pomieścić istniejące treści, przewidzieć miejsce na treści, które pojawią się w przyszłości, oraz pamiętać o tym by pozostawić wolne przestrzenie, tak by strona mogła "oddychać".

Można również zdefiniować tzw. **punkty startowe**, czyli 3 do 5 najważniejszych rzeczy, dla których użytkownicy będą odwiedzali naszą stronę i przedstawić je w sposób graficzny.

Projekt układu logicznego witryny przygotowywany jest **w odcieniach szarości**, co pomaga podczas analizy rozmieszczenia oraz zajmowanej przestrzeni elementów strony. Na tym etapie kolory tła czy fontów nie mają znaczenia. Przygotowujemy tylko odpowiednie miejsce dla treści.

Sztywne, blokowe rozplanowanie przestrzeni nie jest wiążące dla późniejszego projektu graficznego, w którym mogą pojawić się zaokrąglenia i gradienty niwelujące wrażenie układu kolumnowego.

Trzeba zwrócić uwagę na miejsca projektu, które muszą pomieścić treści dodawane w przyszłości. Projekt musi być "rozciągliwy", a w zależności od pojemności treściowej witryny będzie dłuższy, rzadziej szerszy.

Krok 5 - Tworzenie projektu graficznego

Aby stworzyć projekt graficzny, projektant powinien zapoznać się z **systemem identyfikacji wizualnej** firmy, lub w przypadku jego braku, wspólnie z właścicielem witryny dobrać odpowiednie dla konkretnej strony kolory, fonty i inne elementy graficzny (zdjęcia, ikony).

Następnie ustalone, konkretne elementy graficzne zostają nałożone na projekt układu logicznego witryny (krok 4).

układ logiczny witryny
logotyp
kolory
fonty
zdjęcia

+ ikony
---
projekt graficzny

Zazwyczaj strona internetowa powinna posiadać **2 do 3 kolorów wiodących**. Kolory te pojawiają się jako nagłówki, elementy menu, linki w tekście, elementy nawigacyjne. Inne kolory mogą się pojawiać w zdjęciach czy ikonach jednak nie powinny zanadto przytłaczać innych elementów strony.

Krok 6 - Umieszczanie treści

Gotowy projekt graficzny jest "cięty" na mniejsze elementy, tak by każda ikona czy baner był osobnym plikiem graficznym. Pliki te są następnie układane w funkcjonalną stronę internetową (HTML). Teksty tymczasowe zastępowane zostają docelowymi informacjami, które mają znaleźć się na stronie.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In pulvinar venenatis leo. Mauris id odio nec ante posuere lacinia. Suspendisse dui. Aliquam erat volutpat. Maecenas posuere, erat sed convallis aliquet, lacus sapien dictum magna, ac vulputate urna odio at dolor. Maecenas lorem arcu, pharetra id, viverra non, elementum id, nisi.

W dziale "Co możemy dla Ciebie zrobić?" prezentujemy Państwu informacje o usługach serwisowych oferowanych przez naszą firmę, jak również o outsourcingu i rozwiązaniach sieciowych. Jeśli są zainteresowani Państwo stroną internetową, którą można samodzielnie uaktualniać, informacje na ten temat także odnajdą Państwo w tym dziale. W tym samym dziale znajdują się również informacje nt. integracji oprogramowania autorskiego z zewnętrznymi aplikacjami.

Na tym etapie strona zostaje umieszczona na lokalnym serwerze wdrożeniowym, gdzie można podglądać jak z godziny na godzinę na stronie pojawia się coraz więcej treści. W zależności od pojemności witryny i jakości materiałów do umieszczenia proces ten może trwać dłużej bądź krócej. Kiedy na stronie będzie już odpowiednia ilość materiałów, można ją przenieść na serwer docelowy i pokazać całemu światu.