

Podsumowanie.

Elementy inżynierii oprogramowania.

- Zadanie algorytmiczne, specyfikacja
- Podstawy analizy algorytmów: poprawność, złożoność
- Podstawy języka C:
 - typy (int, float, char),
 - instrukcje sterujące (while, do while, if, else),
 - operatory arytmetyczne, logiczne, przypisania
(+ = - * / % && || < > !=)
 - funkcje, zakres zmiennych (lokalne, globalne)
 - tablice jednowymiarowe, łańcuchy, struktury
 - wskaźniki,
strumienie
- Paradygmaty: programowanie proceduralne i strukturalne

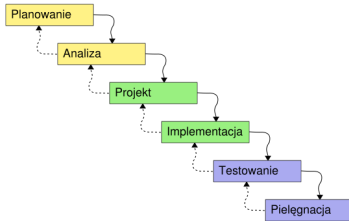
- Dynamiczny przydział pamięci
- Pliki nagłówkowe i źródłowe, biblioteki statyczne, dynamiczne
- Typy danych, m.in.: unie, pola bitowe, typedef
- Tablice wielowymiarowe (tablice tablic)
- Inne złożone struktury danych: listy, kolejki, stosy, drzewa, ...
- Wskaźniki do funkcji, funkcja argumentem funkcji
- Obsługa błędów systemowych
- Operacje na wskaźnikach
- Instrukcje preprocesora i makra
- Masa przydatnych funkcji z biblioteki standardowej
- itd...

INŻYNIERIA OPROGRAMOWANIA

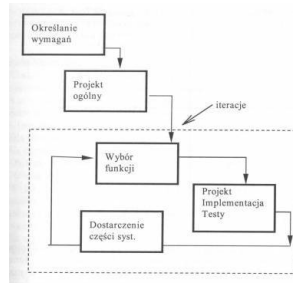
INŻYNIERIA OPROGRAMOWANIA

zajmuje się wszelkimi aspektami produkcji oprogramowania we wszystkich fazach **cyklu życia oprogramowania**

Kaskadowy



Iteracyjny



Źródło: http://pl.wikipedia.org/wiki/Model_kaskadowy,
Ilona Blumke, Inżynieria oprogramowania

- **Analiza wymagań:** opis funkcjonalności i sposób ich realizacji
- **Modelowanie systemu:** organizacja systemu w kategoriach informatycznych. Modele danych, model architektury systemu, model przepływ danych, modele zjawisk.
- **Etap projektowy:** wybór narzędzi, środków technicznych, rozwiązań algorytmicznych.
- **Programowanie:** implementacja, konserwacja kodu, testowanie
- Błędy możliwe na każdym etapie, kumulacja błędów

Hierarchiczne podejście do rozwiązania problemu

- metodyka „zstępująca” (*top-down*), dedukcyjna.
Dekompozycja problemu, od ogółu do szczegółu, podział większego programu na podprogramy
- metodyka „wstępująca” (*bottom-up*), syntetyczna.
Modularne podejście do tworzenia programu. Składanie rozwiązania z istniejących małych narzędzi

ZANIM ZACZNIESZ KODOWAĆ

- Właściwa definicja problemu - zasadnicza część programowania
- Dobór właściwych struktur danych może znacznie uprościć problem
- „Dobry programista jest trochę leniwy” - najpierw pomyśl potem programuj

- Język C bardzo elastyczny, nieostrożne używanie oznacza kłopoty
- Czytelność przede wszystkim
- Spójność stylistyczna - w całym kodzie jednako
- Zrozumiałe i jednolite nazwy zmiennych i funkcji.
Nazwa nie powinna wprowadzać w błąd.
`wypiszwiersz()`, `WypiszWiersz()`, `wypisz_wiersz()`
- Małe i duże litery są rozróżnialne
- Nazwy zmiennych zazwyczaj z małej litery
np. indeksy `i`, `j`, `k`
- Stałe symboliczne i makra dużymi literami
`#define MAX 500`

- Opis funkcji

```
/* Zwraca pozycje wzorca 'w' w lancuchu 't' */  
int strindex(char *t, char *w) { ... }
```

- Nagłówek pliku: autor, data, licencja, wersja
- Nie komentuj oczywistych rzeczy, raczej opis sensu operacji

```
i = i + 1;    /* Zwiększenie licznika o 1 */  
i = i + k;    /* Ustawienie indeksu na ostatni element */
```

- W C lepiej nie używać komentarzy //
- *Jeśli dokumentacja nie powstaje równoległe z programem – nie powstanie nigdy!*
- Dokumentacja techniczna, generatory np. Doxygen

```
1  #include <stdio.h>
2  int nwd(int a,int b){int c;
3  while(b!=0){c=a%b;a=b;b=c;}
4  return a;} int main(){
5  int a,b; printf("Podaj dwie li"
6  "czby calkowite: "); scanf("%d %d",
7  &a,&b); printf("NWD(%d,%d) = %d\n",
8  a,b,nwd(a,b));return 0;}
```

 nwd-balagan.c

- Czytelność przede wszystkim
-  The International Obfuscated C Code Contest

```

1  #include<stdio.h>
2  int nwd(int a,int b)
3  {
4  int c;
5  while (b!=0)
6  {
7  c=a%b;
8  a=b;
9  b=c;
10 }
11 return a;
12 }
13 int main()
14 {
15 int a,b;
16 printf("Podaj dwie liczby calkowite: ");
17 scanf("%d %d",&a,&b);
18 printf("NWD(%d,%d) = %d\n",a,b,nwd(a,b));
19 return 0;
20 }

```

 nwd-balagan2.c

STYL ALLMANA (BSD)

```
int nwd(int a, int b)
{
    int c;
    while (b != 0)
    {
        c = a % b;
        a = b;
        b = c;
    }
    return a;
}
```

STYL K&R (GNU)

```
int nwd(int a, int b)
{
    int c;
    while (b != 0){
        c = a % b;
        a = b;
        b = c;
    }
    return a;
}
```

- Wewnętrzne bloki instrukcji wcięte względem zewnętrznych
- Instrukcje w jednym bloku zaczynają się w tej samej kolumnie
- Nie przesadzaj z długością linii (max. 78 znaków)
- Oddzielaj deklaracje zmiennych od instrukcji lub grupy spójnych instrukcji pustymi liniami
- Długie ciągi instrukcji warto rozbić na kilka linii i otoczyć nawiasami

```
while ( a < b && wpłata(x) != -1 ) {  
    if ( w != NULL ) {  
        a = a - 1 + sin(PI * 2);  
    }  
}
```

- Python - wcięcia elementem składni języka

- Nie więcej niż jedna instrukcja w linii
- Nie deklaruj więcej niż jedną zmienną w linii
- Otwierając nawias od razu go zamknij `{ ([ ] ) }`
- Nawiasy `()` przyklej do nazwy funkcji ale oddziel od instrukcji `if` i `while`

```
void czy_parzysta(int i)
{
    if ( i%2 == 0 ) printf("Parzysta!\n");
}
```

- Zawsze inicjuj zmienne
- Nie używaj magicznych liczb

```
int main()
{
    /* ... */
    if ( x == 42 ) wystrzel_rakiete();
}
```

- Lepiej zdefiniuj stałą

```
#define KONIEC_ODLICZANIA 42
int main()
{
    /* ... */
    if ( x == KONIEC_ODLICZANIA ) wystrzel_rakiete();
}
```

PROGRAMOWANIE PROCEDURALNE I STRUKTURALNE

- **Programowanie proceduralne.**

Unikaj zmiennych globalnych.

Przekazuj dane do funkcji przez argumenty.

- Unikaj długich funkcji.

Dobrze gdy w całości mieści się na ekranie.

- Funkcje powinny realizować jedną operację, jednak w sposób pełny i skończony

- Unikaj powtórzeń kodu - pisz funkcje.

Don't repeat yourself (DRY), once and only once (OAOO)

- **Strukturalność w programowaniu.**

Nie używaj instrukcji skoku

- *Less is more*
- *„Konstruktor wie, iż osiągnął doskonałość, nie wtedy, gdy już nic nie można dodać, lecz wtedy, gdy już nic nie da się ująć”.*
Antoine de Saint-Exupéry
- Jednak nie upraszczaj na siłę

```
void strcpy(char s[], char t[])
{
    int i=0;
    while (t[i] != '\0')
    {
        s[i] = t[i];
        i = i + 1;
    }
    s[i] = '\0';
}
```

```
void strcpy(char *s, char *t)
{
    while (*s++=*t++);
}
```

Jeżeli coś może się nie udać, to się nie uda. Prawo Murphy'ego

SYTUACJE WYJĄTKOWE W FUNKCJACH

- Zanim użyjesz nieznanej funkcji przeczytaj dokumentację!
- Wartość zwracana: NULL, EOF, inna specjalna wartość

```
if ( fopen("plik.txt", "r") != NULL ) { /* OK */ }
```

- Plik nagłówkowy `errno.h` i zmienna `errno` kodująca błąd
- 📖 Kody błędów różne na różnych platformach
- Funkcja `perror()` wypisuje komunikat błędu

```
x = sqrt(-1);  
if( errno == EDOM ) perror("Bład ");
```

ZABEZPIECZANIE SENSOWNOŚCI DANYCH WEJŚCIOWYCH

- niepoprawne dane, zasada GIGO *Garbage In, Garbage Out*
- *idiotoodporność*, przewiduj nawet najgłupsze dane wejściowe
- przepełnienie bufora

```
scanf("%s", tablica);  
gets(tablica);
```

BŁĘDY W APLIKACJACH KONSOLOWYCH

- Standardowy strumień błędów `stderr`

```
fprintf(stderr, "Don\'t panic!!!\n");
```

- Status zakończenia programu. 0 to sukces.

```
void exit(int status);
```

`/* stdlib.h */`

- Poprawność algorytmów: niezmienniki, zbieżność, skończoność
- Testy poprawności implementacji: asercje

`assert( wyrażenie )` `/* assert.h */`

Przerywa program gdy predykat jest fałszywy.

```
1  #include <assert.h>
2  int main()
3  {
4      int i=0, t[10];
5
6      while( i <= 10 )
7      {
8          assert( i < 10 );
9          t[i] = i;
10         i = i + 1;
11     }
12     return 0;
13 }
```

 `assert.c`

- **Debugowanie** - kontrolowane wykonywanie kodu
- **Profilowanie** - pomiary czasu i zajętej pamięci
- Testy funkcjonalności: testy jednostkowe, testowanie pojedynczych funkcji
- Testy aplikacji - programy testujące programy
- Automatyzacja testów
- To, że nie znaleziono błędu, nie znaczy, że go tam nie ma

Najtańszymi, najszybszymi i najsolidniejszymi elementami systemu komputerowego są te, których w nim nie ma”

Gordon Bell, DEC

LITERATURA DODATKOWA

-  Ilona Bluemke, *Inżynieria oprogramowania*
-  Dawid Czermer, Wykład:  Inżynieria oprogramowania
-  Wikipedia:  Indent style,  Asercja
-  Code Smell, <http://c2.com/xp/CodeSmell.html>
-  C Coding Standards,
<http://users.ece.cmu.edu/~eno/coding/CCodingStandard.html>
-  Recommended C Style and Coding Standards, Bell Labs
<http://www.doc.ic.ac.uk/lab/cplus/cstyle.html>

 J. Bentley, *Perłki oprogramowania*, WNT, Warszawa, 2001.

  `errno.h` - system error numbers

 Linux Programmer's Manual: `man errno assert`

 Errno Codes by Platform,
<http://www.ioplex.com/~miallen/errcmp.html>