

Metodyka i Techniki Programowania

Katedra Telekomunikacji, EiT

dr inż. Jarosław Bułat (c)

kwant@agh.edu.pl

Plan prezentacji

- » Operacje dyskowe C i C++
- » Podejście wstępujące/zstępujące

Chciałbym coś zapisać na dysk
fread(...)

Operacje I/O

- » Pliku nie da się używać tak jak tablicy
- » Plik jest poza przestrzenią adresową CPU
(pamięć RAM vs pamięć masowa)
- » Czas dostępu do danych:
 - RAM L1: ~1 ns
 - RAM: ~100 ns
 - SDD: 0.1 ms == 100 us == 100000 ns
 - HDD: 10 ms == 1000 us == 10000000 ns

Operacje I/O

- » Dostęp do pliku wykonywany jest za pośrednictwem funkcji systemu operacyjnego (złożony proces):
 - uzyskanie dostępu do pliku
 - zlecenie przepisania fragmentu pliku do pamięci RAM

Operacje I/O

- » Dostęp do pliku wykonywany jest za pośrednictwem funkcji systemu operacyjnego (złożony proces):
 - uzyskanie dostępu do pliku
 - zlecenie przepisania fragmentu pliku do pamięci RAM
 - czekanie...

Operacje I/O

- » Dostęp do pliku wykonywany jest za pośrednictwem funkcji systemu operacyjnego (złożony proces):
 - uzyskanie dostępu do pliku
 - zlecenie przepisania fragmentu pliku do pamięci RAM
 - czekanie...
 - czekanie...

Operacje I/O

- » Dostęp do pliku wykonywany jest za pośrednictwem funkcji systemu operacyjnego (złożony proces):
 - uzyskanie dostępu do pliku
 - zlecenie przepisania fragmentu pliku do pamięci RAM
 - czekanie...
 - czekanie...
 - w międzyczasie można zrobić coś pożytecznego ;-)

Operacje I/O

- » Dostęp do pliku wykonywany jest za pośrednictwem funkcji systemu operacyjnego (złożony proces):
- uzyskanie dostępu do pliku
 - zlecenie przepisania fragmentu pliku do pamięci RAM
 - czekanie...
 - czekanie...
 - w międzyczasie można zrobić coś pożytecznego ;-)
 - przeczytanie zawartości bufora

Operacje I/O

- » Dostęp do pliku wykonywany jest za pośrednictwem funkcji systemu operacyjnego (złożony proces):
- uzyskanie dostępu do pliku
 - zlecenie przepisania fragmentu pliku do pamięci RAM
 - czekanie...
 - czekanie...
 - w międzyczasie można zrobić coś pożytecznego ;-)
 - przeczytanie zawartości bufora
 - ewentualna modyfikacja i zapis na dysk (jeszcze dłuższe czekanie)

Operacje I/O w C

```
#include <iostream>
#include <stdio.h>
using namespace std;
```

```
int main(){
    char buf[128];
    FILE *file;
    file = fopen("test.txt", "r");

}
```

» Plik nagłówkowy wymaganej biblioteki

» Identyfikator pliku

- to nie jest “wskaźnik na plik”
- to jest wskaźnik na strukturę opisującą plik
- często nazywa się go “fd” (file descriptor)

» Otwarcie pliku do czytania

Operacje I/O w C

```
#include <iostream>
#include <stdio.h>
using namespace std;
```

```
int main(){
    char buf[128];
    FILE *file;
    file = fopen("test.txt", "r");
    fread(buf, 1, sizeof(buf), file);
}
```

- » Plik nagłówkowy wymaganej biblioteki
- » Identyfikator pliku
 - to nie jest “wskaźnik na plik”
 - to jest wskaźnik na strukturę opisującą plik
 - często nazywa się go “fd”
- » Otwarcie pliku do czytania
- » Przepisanie **jednego** bufora o długości **128 bajtów** z **pliku** o identyfikatorze file do **bufora**

Operacje I/O w C

```
#include <iostream>
#include <stdio.h>
using namespace std;
```

```
int main(){
    char buf[128];
    FILE *file;
    file = fopen("test.txt", "r");
    fread(buf, 1, sizeof(buf), file);
    fclose(file);
}
```

- » Plik nagłówkowy wymaganej biblioteki
- » Identyfikator pliku
 - to nie jest “wskaźnik na plik”
 - to jest wskaźnik na strukturę opisującą plik
 - często nazywa się go “fd”
- » Otwarcie pliku do czytania
- » Przepisanie **jednego** bufora o długości **128 bajtów** z **pliku** o identyfikatorze file do **bufora**
- » Zamknięcie pliku (zwolnienie zasobów)

Operacje I/O w C

```
#include <iostream>
#include <stdio.h>
using namespace std;
```

```
int main(){
    char buf[128];
    FILE *file;
    if (!(file = fopen("test.txt", "r"))){
        printf("Could not open file\n");
        return -1;
    }
    size_t size = fread(buf, 1, sizeof(buf), file);
    printf("%zu bytes were read\n", size);
    fclose(file);
    return 0;
}
```

- » Bardziej poprawnie:
- **sprawdzenie** czy udało się otworzyć plik (NULL jeżeli nie)
 - fread zwraca **ile** danych udało się przeczytać danych

Operacje I/O w C

```
FILE *file;  
file = fopen("test.txt", "r");
```

```
fread(buf, 1, 10, file);  
fread(buf, 1, 10, file);
```

- » W pliku jest “wskaźnik” pozycji z której będą czytane dane
- » Po otwarciu jest na początku
- » Pierwsza instrukcja przeczyta 10 bajtów i ustawi ten wskaźnik na bajt 11-sty
- » Druga (kolejna) instrukcja przeczyta bajty o numerach 11...20

Operacje I/O w C

```
FILE *file;  
file = fopen("test.txt", "r");
```

```
fread(buf, 1, 10, file);  
fread(buf, 1, 10, file);
```

- » W pliku jest “wskaźnik” pozycji z której będą czytane dane
- » Po otwarciu jest na początku
- » **Pierwsza instrukcja** przeczyta 10 bajtów i ustawi ten wskaźnik na bajt 11-sty
- » **Druga (kolejna) instrukcja** przeczyta bajty o numerach 11...20

```
hd test.txt
```

```
00000000  50 6f 64 73 74 61 77 79 20 49 6e 66 6f 72 6d 61 |Podstawy Informa|  
00000010  74 79 6b 69 0a 43 2b 2b 20 65 78 61 6d 70 6c 65 |tyki.C++ example|
```


Operacje I/O w C

```
FILE *file;  
file = fopen("test.txt", "r");
```

```
fread(buf, 1, 10, file);  
fread(buf, 1, 10, file);
```

- » W pliku jest “wskaźnik” pozycji z której będą czytane dane
- » Po otwarciu jest na początku
- » **Pierwsza instrukcja** przeczyta 10 bajtów i ustawi ten wskaźnik na bajt 11-sty
- » **Druga (kolejna) instrukcja** przeczyta bajty o numerach 11...20

```
hd test.txt
```

```
00000000  50 6f 64 73 74 61 77 79 20 49 6e 66 6f 72 6d 61 |Podstawy Informa|  
00000010  74 79 6b 69 0a 43 2b 2b 20 65 78 61 6d 70 6c 65 |tyki.C++ example|
```

Operacje I/O w C

```
FILE *file;  
file = fopen("test.txt", "r");
```

```
fread(buf, 1, 10, file);  
fread(buf, 1, 10, file);
```

- » W pliku jest “wskaźnik” pozycji z której będą czytane dane
- » Po otwarciu jest na początku
- » **Pierwsza instrukcja** przeczyta 10 bajtów i ustawi ten wskaźnik na bajt 11-sty
- » **Druga (kolejna) instrukcja** przeczyta bajty o numerach 11...20

```
hd test.txt
```

00000000	50	6f	64	73	74	61	77	79	20	49	6e	66	6f	72	6d	61	Podstawy Informa
00000010	74	79	6b	69	0a	43	2b	2b	20	65	78	61	6d	70	6c	65	tyki.C++ example

Operacje I/O w C

```
FILE *file;  
file = fopen("test.txt", "r");
```

```
fread(buf, 1, 10, file);  
fread(buf, 1, 10, file);
```

- » W pliku jest “wskaźnik” pozycji z której będą czytane dane
- » Po otwarciu jest na początku
- » **Pierwsza instrukcja** przeczyta 10 bajtów i ustawi ten wskaźnik na bajt 11-sty
- » **Druga (kolejna) instrukcja** przeczyta bajty o numerach 11...20

```
hd test.txt
```

00000000	50	6f	64	73	74	61	77	79	20	49	6e	66	6f	72	6d	61	Podstawy Informa
00000010	74	79	6b	69	0a	43	2b	2b	20	65	78	61	6d	70	6c	65	tyki.C++ example

Operacje I/O w C

```
FILE *file;
file = fopen("test.txt", "r");
```

```
fread(buf, 1, 10, file);
fread(buf, 1, 10, file);
```

```
fseek(file, 1, SEEK_SET);
```

» Wskaźnik można przestawić (o jedną pozycję licząc od początku)

hd test.txt

00000000	50	6f	64	73	74	61	77	79	20	49	6e	66	6f	72	6d	61	Podstawy Informa
00000010	74	79	6b	69	0a	43	2b	2b	20	65	78	61	6d	70	6c	65	tyki.C++ example

Operacje I/O w C

```
FILE *file;
file = fopen("test.txt", "r");
```

```
fread(buf, 1, 10, file);
fread(buf, 1, 10, file);
```

```
fseek(file, 1, SEEK_SET);
```

» Wskaźnik można przestawić (o jedną pozycję licząc od początku)

hd test.txt

00000000	50	6f	64	73	74	61	77	79	20	49	6e	66	6f	72	6d	61	Podstawy Informa
00000010	74	79	6b	69	0a	43	2b	2b	20	65	78	61	6d	70	6c	65	tyki.C++ example

Operacje I/O w C

```
FILE *file;
file = fopen("test.txt", "r");
```

```
fread(buf, 1, 10, file);
fread(buf, 1, 10, file);
```

```
fseek(file, 1, SEEK_SET);
fseek(file, 2, SEEK_CUR);
```

- » Wskaźnik można przestawić (o jedną pozycję licząc od początku)
- » Przesuń o 2 pozycje w stosunku do bieżącego miejsca

hd test.txt

00000000	50	6f	64	73	74	61	77	79	20	49	6e	66	6f	72	6d	61	Podstawy Informa
00000010	74	79	6b	69	0a	43	2b	2b	20	65	78	61	6d	70	6c	65	tyki.C++ example

Operacje I/O w C

```
FILE *file;
file = fopen("test.txt", "r");
```

```
fread(buf, 1, 10, file);
fread(buf, 1, 10, file);
```

```
fseek(file, 1, SEEK_SET);
fseek(file, 2, SEEK_CUR);
fseek(file, -10, SEEK_END);
```

- » Wskaźnik można przestawić (o jedną pozycję licząc od początku)
- » Przesuń o 2 pozycje w stosunku do bieżącego miejsca
- » Ustaw na 10-tej pozycji od końca

hd test.txt

00000000	50	6f	64	73	74	61	77	79	20	49	6e	66	6f	72	6d	61	Podstawy Informa
00000010	74	79	6b	69	0a	43	2b	2b	20	65	78	61	6d	70	6c	65	tyki.C++ example

Operacje I/O w C

```
FILE *file;  
file = fopen("test.txt", "r");
```

```
fread(buf, 1, 10, file);  
fread(buf, 1, 10, file);
```

```
fseek(file, 1, SEEK_SET);  
fseek(file, 2, SEEK_CUR);  
fseek(file, -10, SEEK_END);
```

```
long length = ftell(file);
```

- » Wskaźnik można przestawić (o jedną pozycję licząc od początku)
- » Przesuń o 2 pozycje w stosunku do bieżącego miejsca
- » Ustaw na 10-tej pozycji od końca
- » Pobiera aktualną pozycję wskaźnika
 - **long** na x86 to 32 bity!

Operacje I/O w C

```
FILE *file;  
file = fopen("test.txt", "r");
```

```
fread(buf, 1, 10, file);  
fread(buf, 1, 10, file);
```

```
fseek(file, 1, SEEK_SET);  
fseek(file, 2, SEEK_CUR);  
fseek(file, -10, SEEK_END);
```

```
long length = ftell(file);
```

```
fseek(file, 0, SEEK_END);  
long length = ftell(file);
```

- » Wskaźnik można przestawić (o jedną pozycję licząc od początku)
- » Przesuń o 2 pozycje w stosunku do bieżącego miejsca
- » Ustaw na 10-tej pozycji od końca
- » Pobiera aktualną pozycję wskaźnika
 - long na x86 to 32 bity!
- » Najprostszy sposób na określenie długości pliku

Wiele plików równocześnie

Operacje I/O w C

```
FILE *file1;  
file1 = fopen("test1.txt", "r");
```

```
FILE *file2;  
file2 = fopen("test2.txt", "r");
```

```
fread(buf2, 1, 16, file2);  
fread(buf1, 1, 16, file1);
```

» Wiele identyfikatorów - **każdy plik ma swój identyfikator**

Operacje I/O w C

```
FILE *file1;  
file1 = fopen("test1.txt", "r");
```

```
FILE *file2;  
file2 = fopen("test2.txt", "r");
```

```
fread(buf2, 1, 16, file2);  
fread(buf1, 1, 16, file1);
```

» Wiele identyfikatorów - **każdy plik ma swój identyfikator**

- plik 1
- plik 2

Operacje I/O w C

```
FILE *file1;  
file1 = fopen("test1.txt", "r");
```

```
FILE *file2;  
file2 = fopen("test2.txt", "r");
```

```
fread(buf2, 1, 16, file2);  
fread(buf1, 1, 16, file1);
```

```
fseek(file1, 0, SEEK_END);  
fseek(file2, -10, SEEK_END);
```

- » Wiele identyfikatorów - **każdy plik ma swój identyfikator**
 - **plik 1**
 - **plik 2**
- » Kolejność użycia jest dowolna
- » Wskaźnik w obrębie plików jest indywidualny dla każdego z nich - można go przestawiać niezależnie

Zapis

Operacje I/O w C

```
FILE *file;  
file = fopen("test.txt", "w");  
  
fwrite(buf, 1, 16, file);
```

- » **Zapis** analogicznie do czytania, bufor buf, zostnie **zapisany** na dysk rozpoczynając od bieżącej pozycji wskaźnika

Operacje I/O w C

```
FILE *file;  
file = fopen("test.txt", "w");
```

```
fwrite(buf, 1, 16, file);  
fwrite(buf, 1, 16, file);
```

- » Podczas “otwierania” pliku należy podać czy chcemy:
- “r” - czytać
 - “r+” - czytać i zapisywać
 - “w” - zapisywać
 - “w+” - czytać i zapisywać (jeżeli istnieje zostanie skasowany)
 - “a” - dopisywanie (na końcu)
 - “t” - w trybie tekstowym
 - “b” w trybie binarnym

Operacje I/O w C

```
FILE *file;  
file = fopen("test.txt", "w");  
  
fwrite(buf, 1, 16, file);  
fwrite(buf, 1, 16, file);  
  
file1 = fopen("test.txt", "wt");  
  
fprintf(file, "text %d %f", 16, 0.1);
```

- » Zapis jest podobny do czytania
- » W trybie tekstowym użyteczna jest funkcja `fprintf(...)` - jest ona analogiczna do funkcji `printf(...)`

Operacje I/O w C

```
FILE *file;  
file = fopen("test.txt", "w");  
  
fwrite(buf, 1, 16, file);  
fwrite(buf, 1, 16, file);  
  
file1 = fopen("text.txt", "wt");  
  
fprintf(file, "text %d %f\n", 16, 0.1);
```

- » Zapis jest podobny do czytania
- » W trybie tekstowym użyteczna jest funkcja fprintf(...) - jest ona analogiczna do funkcji printf(...)
- » W tym przypadku:
 - plik **text.txt** zostanie otwarty do **zapisu** w **trybie** tekstowym
 - zostanie utworzony nowy plik, jeżeli taki istniał stary zostanie skasowany
 - do pliku zostanie wpisany tekst: **text 16 0.1\n**
 - uwaga na **\n** (unix/windows)

Operacje I/O w C

```
FILE *file;  
file = fopen("test.txt", "w");
```

```
fwrite(buf, 1, 16, file);  
fwrite(buf, 1, 16, file);
```

- » Obie operacje dyskowe są **blokujące** tj. wywołanie funkcji trwa tak długo aż nie zostaną one ukończone

Operacje I/O w C

```
FILE *file;  
file = fopen("test.txt", "w");
```

```
fwrite(buf, 1, 16, file);  
fwrite(buf, 1, 16, file);
```

```
// POSIX async I/O
```

```
aiocb.aio_fildes = fd;  
aiocb.aio_buf = buf;
```

```
aio_write(&aiocb);
```

```
while (aio_error (&aiocb) == EINPROGRESS){;}
```

```
ret = aio_return(&aiocb);
```

- » Obie operacje dyskowe są **blokujące** tj. wywołanie funkcji trwa tak długo aż nie zostaną one ukończone
- » Operacje dyskowe są powolne więc można je robić asynchronicznie (**nieblokująco**):
 - zlecić zapis/odczyt
 - sprawdzać cyklicznie stan...

Operacje I/O w C

```
#include<stdlib.h>
#include<stdio.h>
```

```
int main(){
    FILE *fp, *fw;

    if (!(fp=fopen("in", "r"))){
        exit(1);
    }
    if (!(fw=fopen("out", "w"))){
        exit(2);
    }

    char buf[128];
    while(fread(buf, 1, sizeof(buf), fp)){
        fwrite(buf, 1, sizeof(buf), fw);
    }

    return 0;
}
```

- » **Kopiowanie z pliku do pliku**
- » Otwarcie pliku do odczytu
- » Otwarcie pliku do zapisu
- » **Bufor - nie da się wczytać całego pliku do pamięci!!!**

Operacje I/O w C

```
#include<stdlib.h>
#include<stdio.h>
```

```
int main(){
    FILE *fp, *fw;

    if (!(fp=fopen("in", "r"))){
        exit(1);
    }
    if (!(fw=fopen("out", "w"))){
        exit(2);
    }

    char buf[128];
    while(fread(buf, 1, sizeof(buf), fp)){
        fwrite(buf, 1, sizeof(buf), fw);
    }

    return 0;
}
```

- » **Kopiowanie z pliku do pliku**
- » Otwarcie pliku do odczytu
- » Otwarcie pliku do zapisu
- » **Bufor - nie da się wczytać całego pliku do pamięci!!!**
- » **Odczytanie fragmentu pliku**
- » **Zapisanie tego fragmentu**
- » Funkcja fread(...) zwraca długość przeczytanego fragmentu więc dla ==0 pętla while się zakończy

Operacje I/O w C

```
#include<stdlib.h>
#include<stdio.h>
```

```
int main(){
    FILE *fp, *fw;

    if (!(fp=fopen("in", "r"))){
        exit(1);
    }
    if (!(fw=fopen("out", "w"))){
        exit(2);
    }

    char buf[128];
    while(fread(buf, 1, sizeof(buf), fp)){
        fwrite(buf, 1, sizeof(buf), fw);
    }

    return 0;
}
```

- » **Kopiowanie z pliku do pliku**
- » Otwarcie pliku do odczytu
- » Otwarcie pliku do zapisu
- » **Bufor - nie da się wczytać całego pliku do pamięci!!!**
- » **Odczytanie fragmentu pliku**
- » **Zapisanie tego fragmentu**
- » Funkcja fread(...) zwraca długość przeczytanego fragmentu więc dla ==0 pętla while się zakończy
- » **Jaki błąd semantyczny popełniłem???**

Odczyt zapis w C++ obiektoowo!!!

Operacje I/O w C++

```
#include <iostream>
```

```
#include <fstream>
```

```
using namespace std;
```

```
int main(){
```

```
    ofstream file("txt1.txt");
```

```
    file << "C++ example.\n";
```

```
    file.close();
```

```
}
```

- » Na **obiekcie** klasy **ofstream** wykonywane są wszystkie akcje związane z użyciem pliku:

Operacje I/O w C++

```
#include <iostream>
```

```
#include <fstream>
```

```
using namespace std;
```

```
int main(){  
    ofstream file("txt1.txt");  
    file << "C++ example.\n";  
    file.close();  
}
```

- » Na obiekcie klasy ofstream wykonywane są wszystkie akcje związane z użyciem pliku:
 - **otwarcie**

Operacje I/O w C++

```
#include <iostream>
```

```
#include <fstream>
```

```
using namespace std;
```

```
int main(){  
    ofstream file("txt1.txt");  
    file << "C++ example.\n";  
    file.close();  
}
```

- » Na obiekcie klasy ofstream wykonywane są wszystkie akcje związane z użyciem pliku:
- otwarcie
 - **użycie**

Operacje I/O w C++

```
#include <iostream>
```

```
#include <fstream>
```

```
using namespace std;
```

```
int main(){  
    ofstream file("txt1.txt");  
    file << "C++ example.\n";  
    file.close();  
}
```

- » Na obiekcie klasy ofstream wykonywane są wszystkie akcje związane z użyciem pliku:
- otwarcie
 - użycie (wykorzystanie strumieni)

Operacje I/O w C++

```
#include <iostream>
```

```
#include <fstream>
```

```
using namespace std;
```

```
int main(){  
    ofstream file("txt1.txt");  
    file << "C++ example.\n";  
    file.close();  
}
```

- » Na obiekcie klasy ofstream wykonywane są wszystkie akcje związane z użyciem pliku:
- otwarcie
 - użycie (wykorzystanie strumieni)
 - zamknięcie

Operacje I/O w C++

```
#include <iostream>
#include <fstream>
#include <stdlib.h>
using namespace std;
```

```
int main(){
    ofstream file("c++.txt", ios::out);
    if (!file.is_open()){
        exit(1);
    }
    file << "C++ example.\n";
    file.close();

    return 0;
}
```

» Podczas otwierania mogę podać jakiego rodzaju operację będą wykonywane:

- **ios::out** zapis
- ios::in odczyt
- ios::in | ios::binary odczyt w trybie binarnym (nie txt)

Operacje I/O w C++

```
#include <iostream>
#include <fstream>
#include <stdlib.h>
using namespace std;
```

```
int main(){
    ofstream file("c++.txt", ios::out);
    if (!file.is_open()){
        exit(1);
    }
    file << "C++ example.\n";
    file.close();

    return 0;
}
```

- » Podczas otwierania mogę podać jakiego rodzaju operację będą wykonywane:
 - ios::out zapis
 - ios::in odczyt
 - ios::in | ios::binary odczyt w trybie binarnym (nie txt)
- » Po próbie otwarcia warto sprawdzić czy operacja się udała

Operacje I/O w C++

```
#include <fstream>
using namespace std;

int main(){
    ifstream in("in");

    char buf[128];
    in.read(buf, sizeof(buf));

    in.close();
}
```

- » Czytanie z pliku do bufora
- » Na **strumieniu** można wykonać wiele metod:
 - **czytanie**
 - pisanie
 - manipulacja wskaźnikiem
- » Można przeciążyć operator << aby przesyłać bezpośrednio dane w “moim” typie do pliku (np. json, binary, serializacja, etc...)

kopiowanie C++ vs C

```
#include <fstream>
using namespace std;
```

```
int main(){
    ifstream in("in");
    ofstream out("out");

    out << in.rdbuf();

    in.close();
    out.close();
}
```

```
#include <stdlib.h>
#include <stdio.h>
```

```
int main(){
    FILE *fp = fopen("in", "r");
    FILE *fw = fopen("out", "w");

    char buf[128];
    while(size=fread(buf, 1, sizeof(buf), fp)){
        fwrite(buf, 1, size, fw);
    }

    fclose(fp);
    fclose(fw);
}
```

» C: wymaga uzupełnienia o poprawne kopiowanie “ogona”

Operacje I/O w C++ vs C

```
#include <iostream>
#include <fstream>
using namespace std;
```

```
int main(){
    ofstream file1("test.txt");
    ofstream file2("test.txt");

    file1 << "C++: " << 10;
    file2 << "C++: " << 11;

    file1.close();
    file2.close();
}
```

```
#include <stdio.h>
#include <stdlib.h>
```

```
int main(){
    FILE *file1 = fopen("test1.txt", "w");
    FILE *file2 = fopen("test2.txt", "w");

    fprintf(file1, "text1 %d", 10);
    fprintf(file2, "text2 %d", 11);

    fclose(file1);
    fclose(file2);
}
```

- » C++: dane, akcje i stan procesu połączone w **obiekcie**
- » C: różne funkcje muszą być połączone **identyfikatorem** (file) żeby utrzymać kontekst procesu

podejście wstępujące vs zstępujące

Podójście zstępujące

- » Części funkcyjne programu powinny być **małe** (ułatwia to zrozumienie)
- » Jeżeli taka część jest zbyt duża należy **ją podzielić**
- » Podójście **projektowania zstępującego**:
 - program ma robić 7 rzeczy więc dzielę go na 7 funkcjonalnych części (podprogramów, modułów)
 - pierwszy podprogram ma robić 4 rzeczy więc podzielę go na 4 podpodprogramy
- » W efekcie otrzymuję się duże rozdrobnienie (to dobrze)
- » Każda część powinna być **na tyle duża żeby robić coś konkretnego** i **na tyle mała żeby dało się ją zrozumieć**
- » ang. ***top-down design***

Podjęcie wstępujące

- » **Dopasuj język programowania do problemu**, tak aby część funkcyjna była jak najmniejsza
 - Przeciąż operatory (utwórz nowe - lisp)
 - Użyj abstrakcji
 - Wekstrachuj funkcjonalność
- » **Utwórz “nowy język”** programowania, który będzie idealnie dopasowany do rozwiązywanego problemu
- » Wykonasz dużo pracy przy tworzeniu nowego języka ale za to rozwiążesz problem “w trzech liniijkach”
- » W efekcie **funkcjonalna część programu będzie prostsza** czyli **łatwiejsza w napisaniu, zrozumieniu, rozszerzaniu** i **mniej podatna na błędy**
- » ang. ***bottom-up design*** źródło: <http://esejepg.pl/eseje/progbot.html>

zstępujące vs wstępujące

- » **Cel:** kalkulator na liczbach **zespółonych**
- » **Przykład:**
 - oblicz $x = 1*(a+b)*c$
 - wypisz na ekranie

zstępujące vs wstępujące

```
struct Complex{  
    float re;  
    float im;  
};
```

```
int main(){  
    Complex x,a,b,c;  
    Complex c={1,7};  
    c.re = 2;  
    c.im = 8;  
    c = init(3,9);
```

```
    x = add(a, b);  
    x = mul(1.0, x);  
    x = mul(x, c);  
    my_print(x);  
}
```

» Struktura

» Funkcje:

- init(...)
- add(...)
- mul(...)
- my_print(...)

» Nieczytelny zapis sekcji funkcyjnej:

```
x = mul(mul(1.0, add(a, b)), c);
```

zstępujące vs wstępujące

- » Klasa
- » Przeciążenie operatorów:
 - +
 - *
- » Rozszerzenie funkcjonalności biblioteki:
 - przeciążenie <<
- » Czytelny i prosty zapis części funkcyjnej

`x = 1.0*(a+b)*c;`

```
using namespace std;
```

```
class Complex{  
    operator+();  
    operator*();  
};
```

```
operator<<();
```

```
int main(){  
    Complex x,a,b;  
    Complex c(1,3);  
    c = Complex(2,7);
```

```
    x = 1.0*(a+b)*c;  
    cout << x;
```

```
}
```


zstępujące vs wstępujące

```
struct Complex{  
    float re;  
    float im;  
};  
  
int main(){  
    Complex x,a,b,c;  
    Complex c={1,7};  
    c.re = 2;  
    c.im = 8;  
    c = init(3,9);  
  
    x = add(a, b);  
    x = mul(1.0, x);  
    x = mul(x, c);  
    my_print(x);  
}
```

```
using namespace std;  
  
class Complex{  
    operator+();  
    operator*();  
};  
  
operator<<();  
  
int main(){  
    Complex x,a,b;  
    Complex c(1,3);  
    c = Complex(2,7);  
  
    x = 1.0*(a+b)*c;  
    cout << x;  
}
```

zstępujące vs wstępujące

- » Klasa + przeciążanie operatorów to budowanie języka (dopasowywanie go do problemu)
- » **Część funkcyjna jest prosta**, zrozumiała, ogólna (można ją użyć do innych zadań)
- » Rozszerzenie języka będzie więcej kosztować niż napisanie funkcji realizującej podstawowe działania arytmetyczne, za to mamy dużo “plusów” w części funkcyjnej

Dziękuję