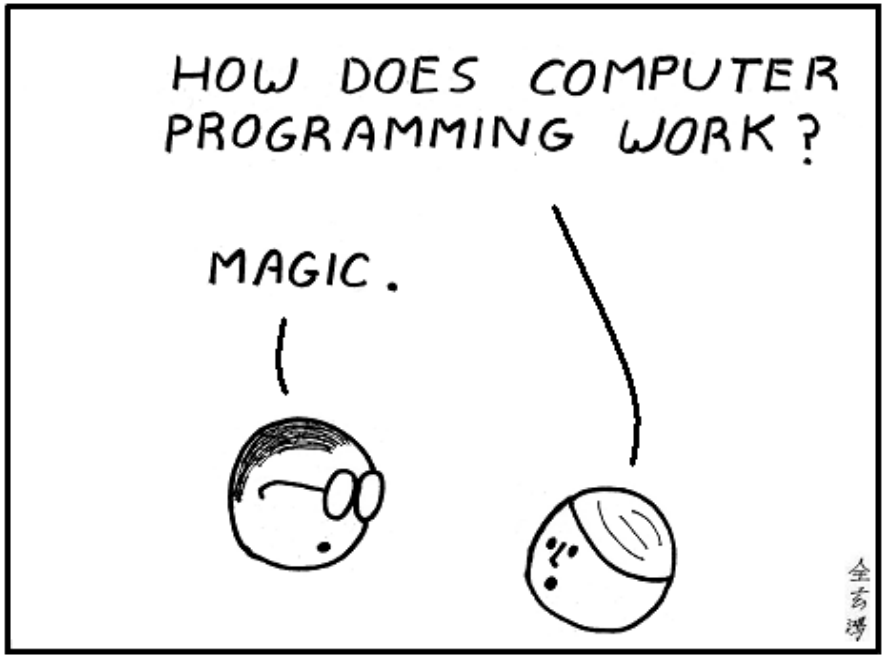
Języki Programowania: C/C++



http://www.saltbox.com/img/under_the_hood.png

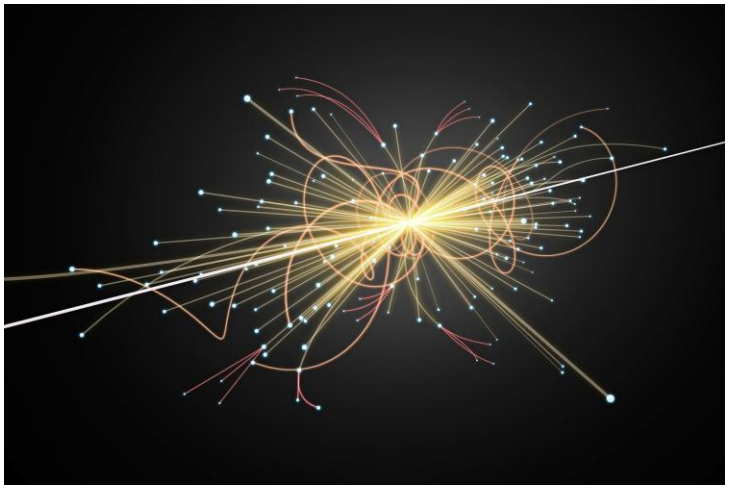
dr hab. inż. Małgorzata Janik

Any sufficiently advanced technology is indistinguishable from magic.

Arthur C. Clarke

O mnie...

- Adiunkt w Zakładzie Fizyki Jądrowej Wydziału Fizyki PW
  - dr hab. inż. Małgorzata Janik
- Kończyłam wydział Fizyki (fizyka komputerowa) i Wydział MiNI (computer science) PW
- Prowadzę badania z Fizyki Cząstek w eksperymencie ALICE na Wielkim Zderzaczu Hadronów w CERN
  - Materia w stanie “plazmy kwarkowo-gluonowej”
  - Korelacje między produkowanymi cząstkami



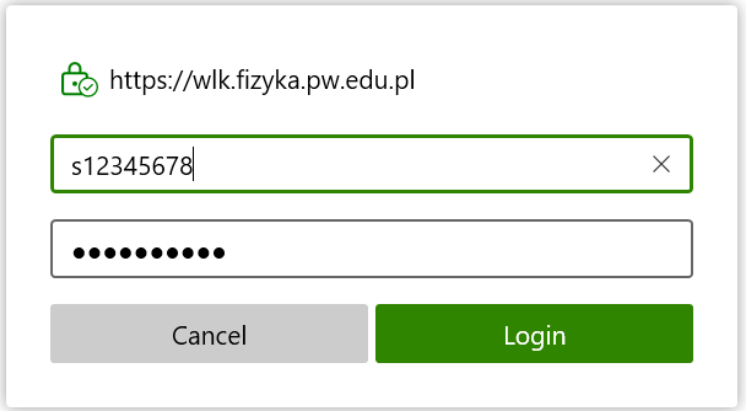
Na czym pracujemy? WLK

Jak się zalogować na komputer? → Windows →
Login: STUD\0USOS_id . Hasło do usosa.

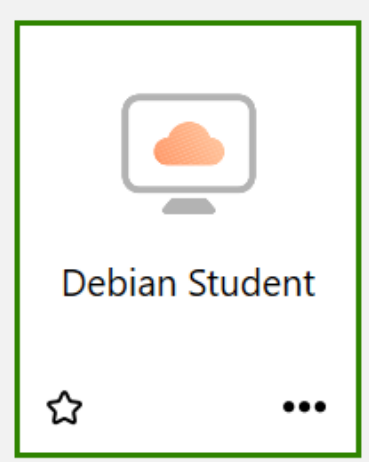
WLK: VMWare Horizon Client → 

→

wlk.fizyka.pw.edu.pl → Login: s0USOS_id →

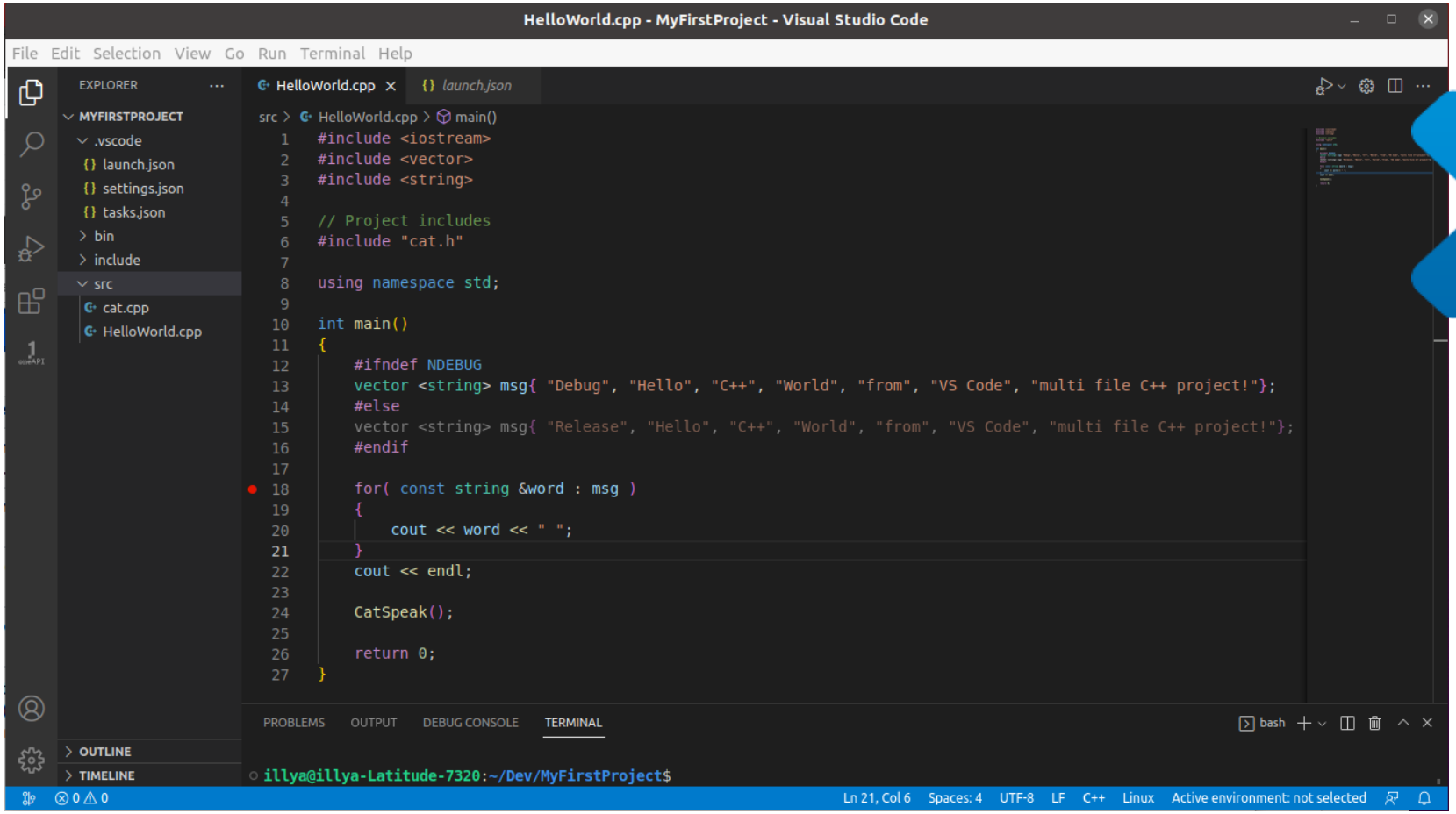


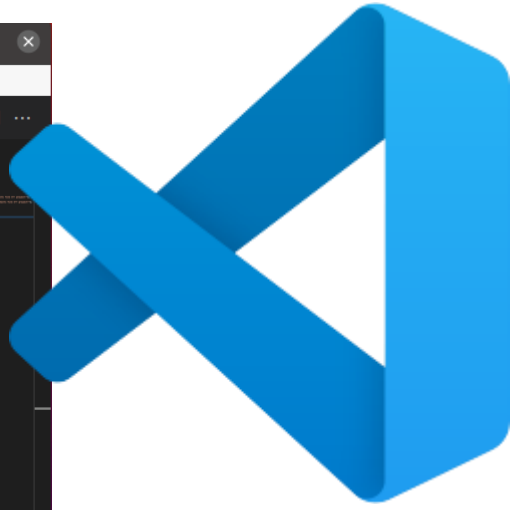
Debian



Na czym pracujemy? VS Code

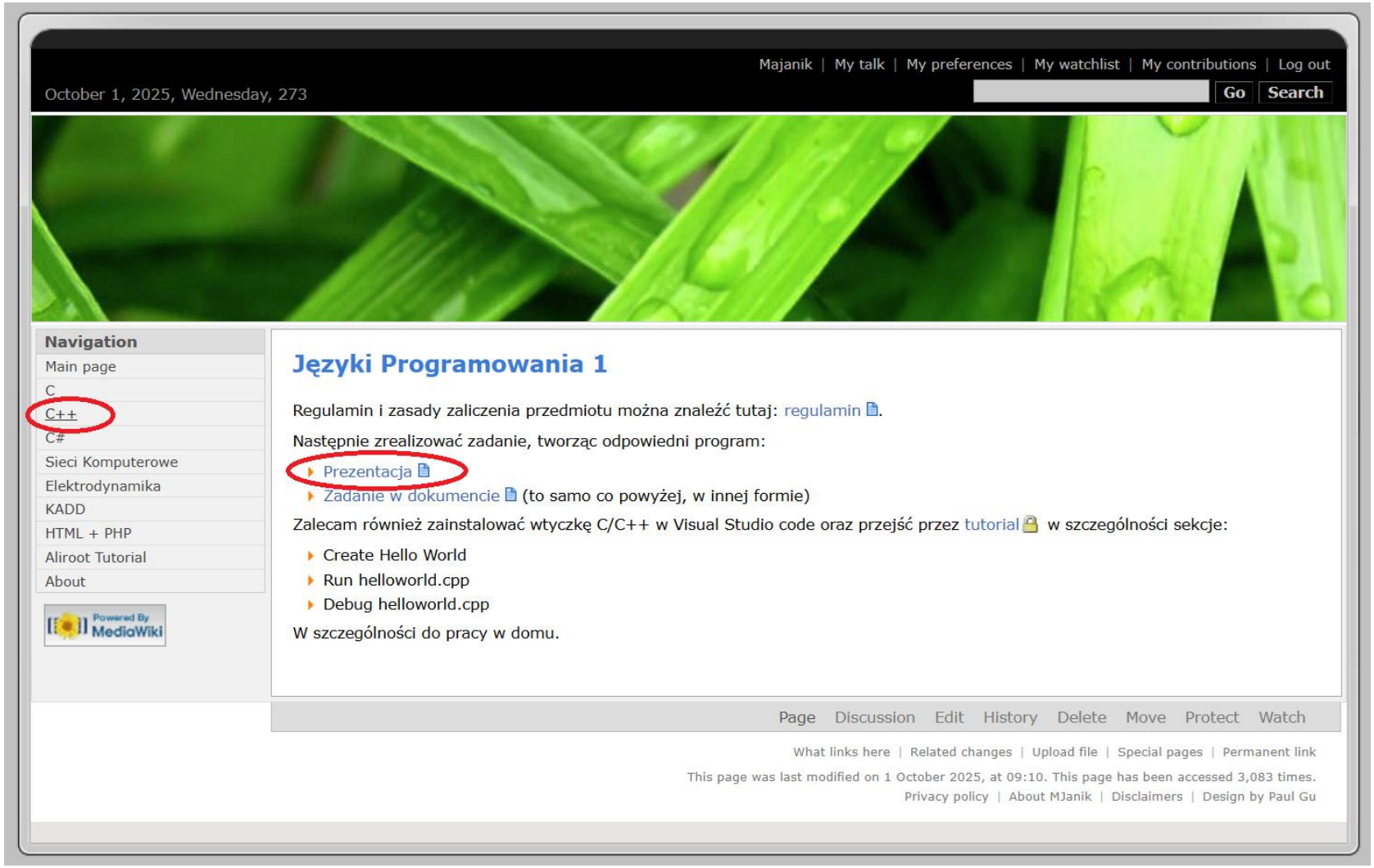
Applications → Programming → Visual Studio Code





https://www.if.pw.edu.pl/~majanik/wiki

Applications → Internet → Firefox



Warunki zaliczenia

- **Wykład – 15 wykładów x 2h co tydzień**
  Nieobowiązkowy. Dla osób uczęszczających na wykłady (możliwa 1 nieobecność), będzie możliwość uzyskanie dodatkowych 10 pkt.
- **Projekt: oddawany na ostatnich zajęciach laboratoryjnych**
  50 pkt
- **Laboratoria: zajęcia punktowane**
  14 x 5 pkt = 70 pkt

- **Warunek zaliczenia:**
  - **Zaliczony project (> 25 pkt)**
  - **Zaliczone zajęcia (> 35 pkt)**

- **Możliwe zaliczenie eksternistyczne.**

Ilość punktów	Ocena
>= 60	3
>= 72	3.5
>= 84	4
>= 96	4.5
>= 108	5

Laboratoria

- **przewidzianych jest 15 zajęć laboratoryjnych (w tym 14 punktowanych, 1 projektowe – oddanie projektów)**
- **obecność jest obowiązkowa (możliwe są maksymalnie 2 nieobecności);**
- **zajęcia trwają 90 minut, odbywają się bez przerwy;**

- **Laboratoria: zajęcia punktowane (14 zajęć)**
- **za każde zadanie można otrzymać 0-5 pkt**
- **w trakcie pisania programu wolno korzystać z napisanych przez siebie programów oraz zasobów Internetu; nie można korzystać z programów innych studentów ani komunikatorów**
- **napisany w trakcie trwania laboratorium program należy oddać na tych samych zajęciach**
- **za skończenie programu po zajęciach (w domu) i przedstawieniu go w kolejnym tygodniu można uzyskać dodatkowy punkt (ale maksymalnie 4 pkt za program)**
- **przynajmniej jeden, a maksymalnie dwa programy będą pisane przez dwa, niekoniecznie następujące po sobie zajęcia;**
- **w przypadku nieobecności studenci są zobowiązani do zrealizowania materiału we własnym zakresie i przedstawienia rozwiązania najdalej 2 tygodnie po nieobecności**

Projekty

- Zajęcia kończą się oddaniem indywidualnego projektu, którego tematyka będzie indywidualnie ustalana z prowadzącym.
- Zadanie projektowe musi zostać oddane przed zakończeniem semestru (maksymalnie w trakcie ostatnich zajęć w semestrze)
- próby niesamodzielnej pracy będą skutkowały niezaliczeniem projektu oraz brakiem możliwości jego poprawy - **niezaliczeniem przedmiotu!**;
- napisany projekt należy przesłać przed końcem semestru na adres e-mailowy prowadzącego / **GitHub**;
- program będzie oceniany w skali 0-50 pkt

Komendy systemu linux

ls (list) - wyświetla zawartość bieżącego katalogu lub katalogu podanego jako parametr.
cd (change directory) - wchodzi do katalogu, no. cd katalog1
cd .. - wchodzi do katalogu wyżej
mkdir (make directory) - do tworzenia katalogów. Przykład: mkdir nazwa_katalogu
cp (copy) - do kopiowania plików i katalogów. Przykłady: cp plik1 plik2
cp -r - kopiuje katalog wraz z zawartością np. cp -r katalog1 katalog2
* - gwiazdka zastępuje dowolny ciąg znaków np.:
cp * alfa/ - kopiuje wszystkie pliki z bieżącego katalogu do katalogu alfa
mv (move) - przenosi plik/pliki, służy też do zmiany nazwy pliku lub katalogu.
mv plik1 plik2 - zmienia nazwę plik1 na plik2
rm (remove) - usuwa pliki. Przykład:
rm plik1 - usuwa plik1
rm * - usuwa wszystkie pliki z bieżącego katalogu (należy używać bardzo ostrożnie - sprawdzić, czy rzeczywiscie chcemy wszystko skasować).
rm -r - usuwa cały katalog razem z zawartością
more - pozwala na przegladanie danych (plików, komunikatów poleceń) ekran po ekranie.
kate plik.txt – uruchamia edytor kate tworząc plik plik.txt
cat - podobnie do polecenia 'more' pokazuje zawartość pliku ale nie zatrzymuje się ekran po ekranie tylko wyświetla od razu całość.

Prosta kompilacja programu – Linux

Plik z kodem źródłowym: program00.c
(pliki z kodem źródłowym języka C powinny mieć rozszerzenie .c)

Plik wynikowy: progam00
(w środowisku linux programy nie posiadają rozszerzenia, lecz wyróżnia je flaga wykonywalności '*x*')

```
gcc program00.c -o program00
```

http://imgs.xkcd.com/comics/compiling.png

Prosta kompilacja programu – Linux

Plik z kodem źródłowym: program00.c
(pliki z kodem źródłowym języka C powinny mieć rozszerzenie .c)

Plik wynikowy: progam00
(w środowisku linux programy nie posiadają rozszerzenia, lecz wyróżnia je flaga wykonywalności '*x*')

```
gcc program00.c  -o program00  -Wall   -pedantic -std=c99
```

Flagi kompilacji:

-Wall	– wyświetla wszystkie ostrzeżenia
-pedantic	– wyświetla niezgodności ze standardem ISO
-std=c99	– stosuj standard C99

Dodatkowo:

-O2	– optymalizacja kompilacji i kodu programu

http://imgs.xkcd.com/comics/compiling.png

Pierwszy program

```
/*****************
 * Jan Kowalski   *
 * 15.03.2013 r.  *
 *****************/



#include <stdio.h>



int main (void)
{

// Wyświetla linijkę tekstu
printf("Moj pierwszy program!");

return 0;  // kończy program
}
```

Komentarz blokowy – dowolny tekst pomiędzy znakami **/*** oraz ***/**

Instrukcja preprocesora – zaczyna się od znaku **#**

Funkcja main() – tutaj zaczyna się sterowanie programem

Instrukcja – linijki na ogół kończą się **średnikiem**

Komentarz – zaczyna się od **//** kończy wraz z końcem linii

Wypisywanie na ekran

```
int main (void)
{

int zmienna = 3;  double zmienna2 = 1.5;
printf("Hello world!\n");
printf("Hello world %d! A to druga: %lf.\n",zmienna,zmienna2);
return 0;

}
```

printf(„Napis\n”); // \n – oznacza znak nowej linii

printf(„%d %f %c”,zmienna_int, zmienna_float, zmienna_char);
//wypisywanie zmiennych

int %d
float %f
double %lf → tylko dwa miejsca po przecinku: %.2lf
char %c
tablica_char %s

(1) „Hello world”

Wypisać na ekran (w terminalu) słowa „Hello World!”

- tworzymy nowy plik tekstowy, nadajemy mu nazwę **hello.c**
- na początku załączamy bibliotekę: `#include <stdio.h>`
- tworzymy funkcję **main**

```
int main()
{
  //tu będziemy wpisywać kod
  return 0;
}
```

- w środku funkcji wypisujemy słowo przy użyciu „printf”
- kompilujemy – przez terminal. W terminalu wpisujemy:

```
gcc hello.c -o hello //jesli hello.c to nasza nazwa pliku
```

Zadanie właściwe: wersja demo

Zadanie polega na napisaniu, skompilowaniu i uruchomieniu prostego kalkulatora który:

1. w pętli pyta jaką operację ma wykonać:
   0 – wyjście, 1 – dodawanie , 2 – odejmowanie)
2. jeśli użytkownik wpisze “0” program się kończy
3. jeśli użytkownik wpisze 1 lub 2 program pyta o dwie liczby, w zależności od znaku wykonuje funkcję dodaj lub odejmij (to powinny być zewnętrzne funkcje przyjmujące 2 argumenty i zwracające wynik).

→ Następnie program wypisuje wynik i wraca do punktu 1.

Zadanie właściwe: wersja finalna

Zadanie polega na napisaniu, skompilowaniu i uruchomieniu prostego kalkulatora który:

1. w pętli pyta jaką operację ma wykonać (+,- lub exit)
2. jeśli użytkownik wpisze “exit” program się kończy
3. jeśli użytkownik wpisze “+” lub “-” program pyta o dwie liczby, w zależności od znaku wykonuje funkcję dodaj lub odejmij (to powinny być zewnętrzne funkcje przyjmujące 2 argumenty i zwracające wynik).

→ Następnie program wypisuje wynik i wraca do punktu 1.

Wczytaj i wypisz

```
int main (void)
{

int zmienna;
scanf(”%d”, &zmienna);

printf(”Wczytana liczba to: %d\n”,zmienna);
return 0;

}
```

scanf(”%d”, &n); // standardowe wejście (klawiatura) wpisz do zmiennej

```
if(scanf(”%d”, &n))
printf(”%d\n”,n);
```

// bez ostrzeżenia

int %d
float %f
double %lf → tylko dwa miejsca po przecinku: %.2lf
char %c
tablica_char %s

Wczytywanie z klawiatury

Wczytywanie liczby całkowitej:

```
1
2 int x;//tworzymy zmienna x, podajemy od razu jej typ
3 printf("Podaj liczbe: ");//pytamy o liczbe
4 scanf("%d", &x);//flaga %d oznacza "wczytaj liczbe calkowita"
```

Wczytywanie ciągu znaków:

```
1 char buf[100];//alokujemy ciag znakow do wczytania customowej
      komendy
2 if (scanf("%99s", buf) != 1) {//wczytanie i sprawdzenie czy sie
      powiodlo
3             return; // błąd wejścia
4 }
5 if (strcmp(buf, "zakoncz") == 0) {//porownanie czy w buf jest "
      zakoncz"
6             printf("Koniec programu.\n");//
7 }
```

#include <string.h> żeby “strcmp” zadziałało.

Instrukcja warunkowa „if”

```
int main (void)
{
int n;
scanf(”%d”, &n);
if (n >= 0)
{
      printf( "Liczba naturalna");
}

return 0;
}
```

Jeśli n większe równe 0

wtedy rób to co w klamrach

Instrukcja warunkowa „if”

```
int main (void)
{
int n;
scanf(”%d”, &n);
if (n >= 0)
{

        printf( "Liczba naturalna.\n");
}
else
{

        printf( "Liczba mniejsza niż 0.\n");
}
return 0;
}
```

wtedy rób to co w kolejnych klamrach

W przeciwnym wypadku

Instrukcja warunkowa „if”

```
int main (void)
{
int n;
scanf(”%d”, &n);
if (n > 0)
{

        printf( "Liczba większa niż 0.\n");
}
else if(n == 0)
{
        printf( "Liczba równa 0.\n");
}
else
{
        printf( "Liczba mniejsza niż 0.\n");
}
return 0;
}
```

- dopisać jeszcze: jeśli użytkownik podał wiek pomiędzy 16 a 17 lat (włącznie) wypisać: ”Już niedługo!”

„i” logiczne to „&&”, czyli np. warunek (a>3 i a <8) to (a>3 && a<8)

Pętla „while”

```
int main (void)
{

int n; int i=1;
scanf(”%d”, &n);
printf(”%d\n”, n);

while (i<=n)
{
   printf(”%d ”,i); // ...
   i++;
}

return 0;
}
```

Pętla „for”

```
int main (void)
{

int n;
scanf(”%d”, &n);
printf(”%d\n”, n);

for (int i=1;i<=n;i++)
{
        printf(”%d ”,i); // ...
}

return 0;
}
```

Pętla „for”: (int i=1;i<=n;i++)

Zaczynając od i równego 0 (int i = 1), do i mniejszego równego n (i<=n), wykonuj raz po raz to co jest w pętli { … }, przy każdej iteracji zwiększając i (i++)

Czyli: n razy wykonaj to, co jest w pętli za każdym razem zwiększając i

Funkcje zewnętrzne

```c
#include <stdio.h>
#include <math.h>

double pierwiastek(double n)
{
  return sqrt(n);
}

int main (void)
{

  printf(“Ile to pierwiastek z 3?\n");
  double pierw = pierwiastek(3);
  printf(“%.2lf”,pierw);

  return 0;  // kończy program
}
```

Funkcja przyjmująca 1 argument
typu double i zwracająca 1 liczbę typu double

Indentacja

```
int main (void)
{
int n;
scanf(”%d”, &n);
if (n > 0)
{
        printf( "Liczba większa niż 0.\n");
}
else if(n == 0)
{
        printf( "Liczba równa 0.\n");
}
else
{
        printf( "Liczba mniejsza niż 0.\n");
}
return 0;
}
```

Brak wcięć nie powoduje błędów kompilacji, jednak prawidłowe używanie wcięć zwiększa czytelność kodu!