Imię i nazwisko	Kierunek	Rok studiów i grupa
Marek Kubicki	Informatyka techniczna ITE	1 rok grupa 5
Data zajęć	Numer i temat sprawozdania	
1.12.2022	Lab 8. – Funkcje.	

Cel:

- Opanowanie podstaw przetwarzania tablic znaków w C.

Przebieg zajęć:

Utworzyłem katalog roboczy lab_8 i skopiowałem do niego pliki prosta_funkcja.c i test_pierwiastka.c
Następnie zacząłem wykonywać polecenia zawarte w pliku prosta_funkcja.c

```
sh-4.2$ pwd
/home/METAL/markubi2/lab_8
sh-4.2$ ls
a.out  prosta_funkcja.c  skr1  skr2  test_pierwiastka.c
```

Utworzyłem zmienną test_main, raz wywołałem funkcję prosta_funkcja [zmieniłem jej definicję aby przyjmowała argument]

```
  int test_main = 5;
  printf("przed wywołaniem prostej funkcji: %d\n", test_main);
  prosta_funkcja(test_main);
```

Oraz wywołałem funkcję prosta_funkcja [zmieniłem jej definicję aby przyjmowała argument]

```
void prosta_funkcja( int test_main );
/…/
void prosta_funkcja( int test_main )
{
/…/
}
```

w funkcji wywołałem zmienną test_fun. Wypisałem zmienną ?1? przed oraz po wywołaniu funkcji a w funkcji dodałem do niej wartość zmiennej test_fun.

```
int test_fun = 10;
  printf("wewnątrz prostej funkcji: zmienna lokalna %d\n", test_fun);
  printf("wewnątrz prostej funkcji: argument przed modyfikacją %d\n",
test_main);
  test_main += test_fun;
  printf("wewnątrz prostej funkcji: argument po modyfikacji %d\n", test_main);
```

Zmiana wartości zmiennej test_main była jedynie lokalna (zawarta w wywołanej funkcji) nie miała wpływu na pierwotną wartość zmiennej.

```
przed wywołaniem prostej funkcji: 5
wewnątrz prostej funkcji: zmienna lokalna 10
wewnątrz prostej funkcji: argument przed modyfikacją 5
wewnątrz prostej funkcji: argument po modyfikacji 15
po powrocie z prostej funkcji: 5
```

Następnie zacząłem wykonywać polecenia zawarte w pliku test_pierwiastka.c

Zadeklarowałem funkcję ?f?

```
double f_pierwiastek( double liczba);
/…/
double f_pierwiastek( double liczba)
{
    double pierwiastek = 1.0;
  double temp;
do{
printf("\n\nFunkcja wywolana jako argument printf: %18.15lf \nFynkcja
wywolana jako argument funkcji: %18.15lf\n", f_pierwiastek( liczba), f_pier-
wiastek( f_pierwiastek( liczba)));

      temp = pierwiastek;
      pierwiastek = 0.5 * (temp + liczba/temp);
    } while(fabs(pierwiastek*pierwiastek - liczba)/liczba > ACCURACY);
return pierwiastek;
}
```

obliczającą oraz zwracającą pierwiastek podanej liczby. Z oryginalnych poleceń printf nie usunąłem niczego poza informacją o liczbie iteracji ponieważ jest ona zamknięta w środku funkcji.

```
Podaj wartość liczby dodatniej (-1 kończy program):
16

liczba do obliczenia pierwiastka: 16.000000000000000
        założona dokładność (względna) obliczania pierwiastka:  0.000010000000000
        pierwiastek liczby: obliczony  4.000000636692939, dokładny  4.000000000000000
błąd bezwzględny:  0.000000636692939, błąd względny:  0.000000159173235
```

Dodałem polecenie printf który jako zmienne przyjmował wywołanie stworzonej przeze mnie funkcji. Pierwsza funkcja jako argument funkcji przyjmowała podaną wcześniej liczbę, a druga wywołanie tej samej funkcji z argumentem jako podana wcześniej liczba wynikiem czego był pierwiastek 4 stopnia.

```
printf("\n\nFunkcja wywolana jako argument printf: %18.15lf \nFynkcja
wywolana jako argument funkcji: %18.15lf\n", f_pierwiastek( liczba), f_pier-
wiastek( f_pierwiastek( liczba)));
```

```
Funkcja wywolana jako argument printf:  4.000000636692939
Fynkcja wywolana jako argument funkcji:  2.000000252095688
```

Wnioski

Funkcje ułatwiają pisanie programu pozwalając na uproszczenie konstrukcji kodu. Za pomocą jednej sprytnie napisanej funkcji można osiągnąć wiele celów w zakresie działania programu. Funkcje umożliwiają łatwiejsze udostępnianie kodu, będąc tak naprawdę jego wyciętym fragmentem. Dzięki funkcjom można znacząco zmniejszyć rozmiar napisanego kodu.