Podstawy programowania
Materiały dydaktyczne do laboratorium

dr inż. Krzysztof Bruniecki

18 października 2016

Zadania domowe

1 Podprogramy i funkcje - wprowadzenie

Programy pisane do tej pory, charakteryzowały się tym, że całość logiki zawierała się w funkcji `main`. W sytuacji gdy program jest rozbudowany, zgodnie z dobrą praktyką, należy go podzielić na podprogramy. W języku `C` można w tym celu wykorzystać funkcje. Funkcje stanowią fragmenty kodu realizujące z góry określone zadania. Ważnym powodem do stosowania funkcji jest również unikanie powtarzania takiego samego kodu źródłowego w wielu miejscach programu. Funkcje mogą przetwarzać dane podawane im jako argumenty. Funkcje mogą zwracać wyniki przy użyciu instrukcji `return`. Funkcja `main` to najprawdziwsza funkcja, taka sama jak pozostałe których dotyczyć będzie to laboratorium. Jest ona szczególna tylko dlatego, że zgodnie ze standardem stanowi punkt wejścia do naszego programu.

W `C` Występują funkcje z jakimi mieliśmy już do czynienia — z biblioteki podstawowej języka `C` znamy m. in. funkcje:

```
int printf(...)        //wypisywanie na ekran
int scanf(...)         //czytanie z klawiatury
double sqrt(double)    //pierwiastkowanie
```

W bibliotekach standardowych występuje wiele pożytecznych funkcji. Dzięki ich znajomości programista może znacznie szybciej implementować programy. W celach treningu można spróbować zaimplementować własne wersje takich funkcji.

Należy zwrócić uwagę że często wykorzystywane `cin` oraz `cout` to nie są funkcje. Są to obiekty, ale o tym później. W szczególności, wyrażenie `<<` NIE jest alternatywną formą wywołania funkcji.

Wracając do funkcji `main`, w jej przypadku argumenty przekazywane są do niej przez system operacyjny. W ciele funkcji `main` możemy z nich korzystać w taki sam sposób jak w innych funkcjach. Mimo że do tej pory wykorzystywaliśmy wariant funkcji `main` bez parametrów, to istnieje również wariant przyjmujący dwa parametry (między innymi). Ilustruje to poniższy przykład.

Przykład 1. Parametry funkcji `main`

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

int main(int argc, char* argv[])
{
  cout << "argc:\t\t" << argc << endl;
  for(int i=0;i<argc;i++) {
    cout<<"argv[" <<i<< "]:\t"<<argv[i]<<endl;
  }
  return 0;
}
```

Zadanie 1. Wywołaj program z powyższego przykładu z argumentami podawanymi z linii poleceń[1] (np. `program.exe argument1 imie nazwisko`).

Instrukcje jakie ma wykonać funkcja są zapisywane w jej definicji (ciele funkcji). Definicja składa się z pierwszej linii zawierającej nazwę funkcji oraz instrukcji zawartych w nawiasach klamrowych. Ogólną składnię funkcji ilustruje poniższy, uproszczony przykład.

Przykład 2. Składnia funkcji umożliwiająca zwrócenie wartości

```
<typ> nazwa_funkcji( [ <lista-parametrów> ] )
{
  // instrukcje do wykonania (ciało funkcji)
  return (<wyrażenie-zwracane>);
}
```

Podana wcześniej funkcja `main` jest zgodna z tą ogólną postacią. Zawiera listę parametrów, typ zwracany, instrukcje do wykonania i zwraca wartość (zero). Wartość ta jest interpretowana przez system operacyjny.

2 Łańcuchy znaków

Bardziej przydatny przykład funkcji zwracającej wartość (gdyż wartość może być wykorzystana przez nasz program) przedstawiony jest poniżej. Przykład prezentuje funkcję przyjmującą jako parametr łańcuch znaków (tablicę znaków) oraz zwracającą jego długość.

Przykład 3. Obliczanie długości łańcucha znaków

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

int dlugosc(char lan[])
{
  int dlugosc=0;
  while(lan[dlugosc]!=NULL) dlugosc++;
```

[1] Najłatwiej to wykonać w konsoli, chociaż można również odpowiednio skonfigurować środowisko programistyczne

```
  return dlugosc;
}
int main(int argc, char* argv[])
{
  cout << "argc:\t\t" << argc << endl;
  for(int i=0;i<argc;i++) {
    int d = dlugosc(argv[i]);
    cout<<"dlugosc(argv["<<i<<"]):\t"<<d<<endl;
  }
  return 0;
}
```

W przypadku gdy nie potrzebujemy aby funkcja zwracała wartości możemy podać typ `void` (nieokreślony) jako typ zwracany[2]. Ilustruje to poniższy przykład.

Przykład 4. Składnia funkcji nie umożliwiająca zwrócenia wartości

```
void nazwa_funkcji ( [ <lista-parametrów> ] )
{
  // instrukcje do wykonania (ciało funkcji)
  return;
  //w tym przypadku instrukcja return nie zwraca wartości,
  //ale powoduje zakończenie i wyjście z funkcji
  cout << "To jest tzw. nieosiągalny fragment kodu";
}
```

Zadanie 2. Zaimplementuj funkcję nie zwracającą wartości, która wypisuje podany łańcuch znaków podaną liczbę razy. Lista parametrów powinna zawierać dwa elementy oddzielone przecinkiem. Sprawdź jej użycie.

3 Nagłówek i ciało funkcji

W języku `C` w przypadku funkcji, pod pojęciem definicji rozumiemy ciało funkcji, czyli jej implementację. Poza definicją funkcji, w języku `C`, może wystąpić również deklaracja. Deklaracja może być przykładowo umieszczona w pliku nagłówkowym (*.h) (alternatywnie deklarację nazywa się nagłówiem funkcji). Może być również umieszczona w pliku z kodem źródłowym (*.c). Nagłówek funkcji/deklaracja zawiera jedynie nazwę, typ zwracany i parametry funkcji (pojedynczy parametr to typ oraz nazwa). Nagłówek/deklaracja to w zasadzie pierwsza linia funkcji, z tym że zakończona jest średnikiem. Wystąpienie deklaracji funkcji jest wymagane przez kompilator przed miejscem wywołania takiej funkcji, chyba że PRZED wywołaniem danej funkcji kompilator napotka pełną definicję (czyli należy umieścić definicję LUB deklarację funkcji w kodzie źródłowym przed jej użyciem). Przykładowo - gdy zdefiniujemy ciało nowej funkcji przed funkcją `main` nie potrzebujemy wówczas osobnej deklaracji. Gdy jednak nie chcemy umieszczać całej implementacji funkcji przed funkcją `main` powinniśmy skorzystać z nagłówka.

Poniższy przykład prezentuje zastosowanie nagłówka i przeniesienie definicji funkcji za funkcję `main`.

[2]W `Pascalu` funkcje nie zwracające wartości nazywane były procedurami

Przykład 5. Maksimum dwóch liczb - nagłówek funkcji

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

int max2(int a,int b);  //nagłówek funkcji max2

int main(int argc, char* argv[])
{
  int k,l,max;
  cin >> k;
  cin >> l;
  max = max2(k,l);
  cout << max <<endl;
  return 0;
}
int max2(int a,int b)
{
  if(a>b)
    return a;
  return b;
}
```

Zadanie 3. Zaimplementuj funkcje `max3`, `max4`, `max5`. W ich implementacji wywoływać funkcję `max2`. Dodać nagłówki tych funkcji.

Zadanie 4. Sprawdź, czy ważna jest kolejność nagłówków w przypadku gdy funkcja korzysta z drugiej (to że ważna jest kolejność implementacji — gdy nie stosujemy nagłówków — już wiemy).

Poniższy przykład prezentuje funkcję obliczającą specyficzną odmianę silni, oznaczaną często podwójnym wykrzyknikiem ($n!!$).

Przykład 6. Implementacja ($n!!$)

```
#include<iostream>
using namespace std;
int pewna_silnia(int n)
{
  int wynik = 1;
  for(int i=n;i>0;i-=2)
    wynik*=i;
  return wynik;
}
int main()
{
  int n,s;
  cout << "Podaj n: ";
  cin >> n;
  s = pewna_silnia(n);
  cout << "Wynik: " << s;
  return 0;
```

```
}
```

Zadanie 5. Przeanalizuj poniższy program. Słyszałeś o liczbach Fibonacciego?

```
#include<iostream>
using namespace std;
int fibo(unsigned int n){
  int f0=0,f1=1,wynik;
  if(n==0) return f0;
  if(n==1) return f1;
  for(int i=2;i<=n;i++){
    wynik = f0+f1;
    f0 = f1;
    f1 = wynik;
  }
  return wynik;
}
int main(int argc, char* argv[])
{
  unsigned int nr;
  cin >> nr;
  cout << fibo(nr)<<endl;
}
```

Zadanie 6. Napisz funkcję, która oblicza n-tą liczbę ciągu podobnego do Fibonacciego: $F^*_{n+3} = a \cdot F^*_{n+2} + b \cdot F^*_n$, $F^*_0 = 0$, $F^*_1 = 0$, $F^*_2 = 1$.

```
double f_star(int n,double a,double b);
```

Zadanie 7. Uruchom i przeanalizuj co robi zagadkowa funkcja poniżej.

Przykład 7. Zagadkowa funkcja

```
#include<iostream>
#include<cmath>
using namespace std;
void zagadka(int set_size)
{
  int sub_num = (1 << set_size);
  for(int i=0;i<sub_num;i++)
  {
    cout << "{";
    for(int j=1,el=0;j<sub_num;j*=2,el++)
      if(j & i) cout << el <<" ";
    cout << "}" << endl;
  }
}
int main()
{
  int n;
  cout << "Podaj n: ";
```

```
  cin >> n;
  zagadka(n);
  return 0;
}
```

4 Quiz

1. Co się stanie i co zostanie wypisane gdy uruchomisz poniższy program?

```
#include <iostream>
using namespace std;
int main(int argc, char* argv[]) {
  char t[] = "AB";
  cout << argc << endl;
  cout << t << endl;
  t[1] = t[0];
  if (argc >= 0) main(-1, NULL);
  return 0;
}
```

2. Co zostanie wypisane gdy uruchomisz poniższy program? Jak sensowniej nazwać użytą funkcję? Dlaczego?

```
#include <iostream>
using namespace std;
int xent(int a) {
  return ++a;
}
int main() {
  int a = 0;
  cout << xent(a) << endl;
  cout << xent(xent(a)) << endl;
  cout << a << endl;
  return 0;
}
```

3. Co zostanie wypisane gdy uruchomisz poniższy program? Jak sensowniej nazwać użytą funkcję? Dlaczego?

```
#include <iostream>
using namespace std;
int tunoc() {
  static int a;
  a++;
  return a;
}
int main() {
  int a = 100;
  cout << tunoc() << endl;
```

```
  cout << tunoc() << endl;
  cout << a << endl;
  return 0;
}
```

4. Co zostanie wypisane gdy uruchomisz poniższy program? Jak sensowniej nazwać użytą funkcję? Dlaczego?

```
#include <iostream>
using namespace std;
void tuc(char t[], int a) {
  t[a] = '\0';
}
int main() {
  char s[15] = "funkcjeWC";
  cout << sizeof s << endl;
  tuc(s, 3);
  cout << s[0] << endl;
  cout << sizeof s << endl;
  tuc(s, 5);
  cout << s << endl;
  return 0;
}
```

5. Co zostanie wypisane gdy uruchomisz poniższy program? Jak sensowniej nazwać użyte funkcje? Dlaczego?

```
#include <iostream>
using namespace std;
void waps(char t[], int i, int j) {
  t[i] ^= t[j];
  t[j] ^= t[i];
  t[i] ^= t[j];
}
void vin(char t[], int a) {
  int i = 0;
  while (i<a/2) {
    waps(t, i, a - i - 1);
    i++;
  }
}
int main() {
  char t[] = "A NA KEI ZDUNA FELA";
  vin(t, (sizeof t) - 1);
  cout << t << endl;
  return 0;
}
```

6. Co zostanie wypisane gdy uruchomisz poniższy program? Jak sensowniej nazwać użyte funkcje? Dlaczego?

```
#include <iostream>
using namespace std;
int* cnierp(int *a) {
  (*a)++;
  return a;
}
int cnitsop(int *a) {
  return (*a)++;
}
int main() {
  int a = 10;
  cout << *cnierp(&a) << endl;
  cout << cnitsop(&a) << endl;
  cout << a << endl;
  return 0;
}
```

Odpowiedzi do quizu

1. Program wypisze sekwencję:
   (wartość uzależniona od liczby parametrów linii poleceń)
   AB
   -1
   AB

   Zwróć uwagę, że zagnieżdżone (rekurencyjne) wywołanie funkcji `main` ma własną lokalną kopię tablicy znaków.

2. Funkcję można nazwać `next` bo zwraca następną wartość. Program wypisze sekwencję:
   1
   2
   0

3. Funkcję można nazwać `count` bo zlicza liczbę wywołań niej samej. Program wypisze sekwencję:
   1
   2
   100

4. Funkcję można nazwać `cut` bo obcina "c-łańcuch" znaków. Program wypisze sekwencję:
   15
   f
   15
   fun

5. Funkcje można nazwać `swap` oraz `inv` bo pierwsza zamienia miejscami dwa znaki w tablicy, a druga odwraca kolejność znaków w tablicy. Program wypisze sekwencję:

`ALEF ANUDZ IEK AN A`
którą można przeczytać jako: `ALE FAN U DZIEKANA` :)
Czy potrafisz uzasadnić jak działa funkcja `swap`?

6. Funkcje można nazwać `preinc` oraz `postinc` bo pierwsza to odpowiednik preinkrementacji zmiennej, a druga postinkrementacji. Program wypisze sekwencję:
11
11
12
Czy potrafisz uzasadnić dlaczego `preinc` zwraca poprzez adres a `postinc` poprzez wartość?

5 Dodatkowe zadania do samodzielnego rozwiązania

Zadanie 8. Napisz funkcję, która oblicza współczynnik dwumianu Newtona $\binom{n}{k}$. Skorzystaj z zależności $\binom{n}{k} = \frac{n!}{k!(n-k)!}$.

```
int dwumian(int n,int k);
```

Zadanie 9. Zaimplementuj i przetestuj funkcję umożliwiającą potęgowanie liczb zmiennoprzecinkowych[3], o następującym nagłówku:

```
float potega(float podstawa, unsigned int wykladnik);
```

Zadanie 10. Napisz funkcję wyznaczającą sumę potęg (*wyk* jest wykładnikiem), *il* kolejnych liczb całkowitych począwszy od liczby *pocz* i zwracającą tę wartość.

```
int suma(unsigned int wyk, unsigned int il, int pocz);
```

Użyj funkcję potęga z poprzedniego zadania.

Zadanie 11. Zaimplementuj funkcję zamieniającą małe litery na duże w łańcuchu znaków podawanym jako parametr.

Zadanie 12. Napisz funkcję, która kopiuje łańcuch znaków z jednej tablicy do drugiej. Zastosuj funkcję do argumentów przekazywanych z linii poleceń (parametry funkcji `main`). Dzięki temu uzyskasz w programie ich kopię.

```
void kopiuj(char cel[],char zrodlo[]);
```

Zadanie 13. Zmodyfikuj funkcję `zagadka`. Niech wypisuje wyłącznie zbiory składające się z dokładnie określonej liczby elementów.

```
void zagadka_mod(int set_size,int subset_size);
```

Zadanie 14**. Napisać funkcję, która generuje k-elementowe kombinacje z powtórzeniami[4] zbioru n-elementowego.

Zadanie 15**. Napisać funkcję generującą wszystkie permutacje zbioru n-elementowego.

[3] Potęgowanie zrealizować z wykorzystaniem dowolnie wybranej pętli, można przyjąć że wykładnik jest liczbą naturalną (wraz z zerem).

[4] `http://pl.wikipedia.org/wiki/Kombinacja_z_powt%C3%B3rzeniami`

Zadanie 16. Napisz funkcję obliczającą pojedynczą linię w trójkącie Pascala [5].

```
int pas_tr(int numer_l, int liczby[], int rozm_tab);
```

Zakładamy że funkcja wypełni przekazaną tablicę. Zabezpiecz funkcję przed przekroczeniem rozmiaru. Skorzystać z metody "'linia po linii"'. Skorzystać z zależności: $\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$.

Zadanie 17. Napisz funkcję, która liczy ilość wystąpień danej litery w łańcuchu znaków.

```
int ilosc_wystapien(char lan[], char litera);
```

Zadanie 18. Napisz funkcję, która liczy ilość wystąpień łańcucha znaków w łańcuchu znaków.

```
int ilosc_powtorzen(char lan_zrodlo[], char lan_wzorzec[]);
```

Zadanie 19. Napisz funkcję, która znajduje element maksymalny w tablicy i zwraca jego indeks.

```
int max_tab(int tab[], int ilosc);
```

Zadanie 20*. Napisz funkcję, która pobiera od użytkownika napis za pomocą tablicy znaków i w tej samej tablicy zwraca jeden z poprzednio podanych łańcuchów (losowy). Skorzystaj ze statycznej tablicy do przechowywania łańcuchów. Ogranicz rozmiar statycznej tablicy zarówno względem ilości przechowywanych tekstów jak i ich rozmiaru. Zastosuj nagłówek postaci:

```
void zapisz_i_pobierz(char lan[],int rozmiar_bufora_wej);
```

skorzystaj z bibliotecznej funkcji `strcpy`.

Zadanie 21*. Napisz funkcję, która znajduje n największych elementów w tablicy i zapisuje je w tablicy pomocniczej.

```
int max_from_tab(const int tab[], int tab_size,int result_size, int result[]);
```

[5] `http://pl.wikipedia.org/wiki/Tr%C3%B3jk%C4%85t_Pascala`

Praca na zajęciach

1 Samodzielna implementacja funkcji matematycznych

W standardowej bibliotece matematycznej występują przydatne funkcje pomocne dla realizacji różnych algorytmów m.in. funkcje trygonometryczne przydatne chciażby w grafice komputerowej. Można je również zaimplementować samodzielnie z użyciem podstawowych operacji arytmetycznych. Ilustruje to kolejne zadanie.

Zadanie 22. Napisz własną implementację funkcji cosinus korzystając z rozwinięcia w szereg. Porównaj jej dokładność w zależności od ilości sumowanych wyrazów szeregu z funkcją biblioteczną `cos()` [6].

$$\cos x = \sum_{i=0}^{\infty} (-1)^i \frac{x^{2i}}{(2i)!} \tag{1}$$

2 Losowość w programach, funkcje modyfikujące łańcuchy znaków, zmienne statyczne...

Przypomnijmy jak generować liczby pseudolosowe... przyda się to w kolejnym zadaniu.

Przykład 8. liczby losowe i funkcja `rand`

```
#include<iostream>
#include <stdlib.h>      /* srand, rand */
#include <time.h>        /* time */
using namespace std;
int main()
{
  const int max_rand = 10;
  const int it_num = 10;
  cout << "Bez srand():"<<endl;
  for(int i=0;i<it_num;i++)
    cout << rand() % max_rand << endl;
  cout << "Ze srand():"<<endl;
  srand (time(NULL));
  for(int i=0;i<it_num;i++)
    cout << rand() % max_rand << endl;
}
```

[6] `#include<math.h>`

Zadanie 23. Napisz funkcję, która losowo zamienia litery (wykonuje losową permutację) w łańcuchu znaków, który podawany jest jako parametr. Przetestuj program na argumentach przekazywanych w linii poleceń. Niech program ciąg znaków po permutacji wypisuje na ekran. Zastosuj i popraw poniższy szablon.

```
#include<iostream>
using namespace std;
int main(int argc, char * argv[])
{
  permutuj(argv[0]);
  cout << argv[0];
}
void permutuj(char wyraz[])
{
  //ta permutacja jest za mało losowa
  char temp;
  temp = wyraz[2];
  wyraz[2] = wyraz[5];
  wyraz[5] = temp;
  return;
}
```

W tym przykładzie można zauważyć, że tablice przekazywane do funkcji można modyfikować. Ewentualne modyfikacje w tablicach są widoczne poza funkcją.

Liczby pseudolosowe można generować na wiele sposobów. Jednym z prostszych jest obliczanie kolejnych wyrazów ciągu:

$$\begin{array}{rcl} a_1 & = & p \\ a_n & = & (a_{n-1} * q) \pmod r \end{array} \tag{2}$$

gdzie p, q, r są pewnymi ustalonymi liczbami pierwszymi.

Zadanie 24. Zastąp biblioteczny mechanizm generowania liczb losowych własną funkcją o nagłówku:

```
int my_rand();
```

Aby nie powtarzać tej samej liczby losowej przy kolejnych wywołaniach skorzystaj z lokalnej zmiennej statycznej, aby funkcja przechowywała swój stan. Skorzystaj z liczb $p = 11$, $q = 4909$, $r = 7919$.

Zadanie 25. Uwzględnij w swojej "bibliotece" liczb losowych mechanizm inicjalizacji generatora. Użyj statycznych zmiennych globalnych. Czym różnią się statyczne zmienne globalne od globalnych?

```
int my_srand(unsigned int init_value);
```

3 Przekazywanie do funkcji tablic wielowymiarowych

Zadanie 26. Przeanalizuj poniższy program. Przedstawia on przykład przekazywania dwuwymiarowych tablic do funkcji. Zwróć uwagę, że w nagłówku funkcji musi być częściowo określony rozmiar tablicy.

```
#include<iostream>
using namespace std;
#define MAX_DLUGOSC 16
int wypisz_slowa(char slowa[][MAX_DLUGOSC], int il_slow)
{
  int i;
  for(i=0;i<il_slow;i++)
    cout <<i<<": "<< slowa[i]<<endl;
  return 0;
}

int main()
{ char tab[] = "Slowo";
  char tab2[][MAX_DLUGOSC] = {"Pierwsze","Drugie","Trzecie"};
  wypisz_slowa(tab2, sizeof(tab2)/sizeof(tab2[0]));
}
```

Zadanie 27. Zmodyfikuj funkcję `zagadka`. Niech zamiast liczb wyświetlają się podzbiory słów ze zbioru wybranego przez Ciebie. Wybrany zbiór słów przekazywać do funkcji w postaci tablicy dwuwymiarowej.