Język C – zajęcia nr 8

Pliki źródłowe (przypomnienie z C01)

Tekst programu w języku C może być zapisany **w jednym lub większej liczbie plików źródłowych.** Kompilacja każdego pliku źródłowego (modułu źródłowego) przebiega oddzielnie. Wszystkie uzyskane podczas kompilacji fragmenty wynikowe (moduły półskompilowane) są łączone ze sobą oraz innymi podobnymi fragmentami pochodzącymi z bibliotek, co prowadzi do utworzenia modułu wykonywalnego.

Niezależna kompilacja plików źródłowych:

kotek1.c $\rightarrow$ kotek1.obj

kotek2.c $\rightarrow$ kotek2.obj

kotek3.c $\rightarrow$ kotek3.obj

Konsolidacja modułów półskompilowanych i bibliotek:

kotek1.obj + kotek2.obj + kotek3.obj + bib_1.lib + ... + bib_n.lib $\rightarrow$ kotek.exe

Wiązanie nazwy obiektu

Po kompilacji programu z każdą nazwą obiektu (np. zmiennej, stałej, funkcji, …) musi być związany jednoznacznie obiekt, który ta nazwa oznacza. Pewna nazwa może występować w wielu różnych miejscach pliku źródłowego, a nawet w różnych plikach źródłowych. Rodzaj **wiązania nazwy** określa sposób, w jaki danej nazwie przypisywany jest obiekt.

Rodzaje wiązań:

- **Nazwa o wiązaniu zewnętrznym** - we wszystkich plikach źródłowych tworzących program oznacza ten sam obiekt. Tylko w jednym miejscu występuje definicja obiektu, w pozostałych jedynie deklaracje. Obszar widzialności takiej nazwy rozciąga się na inne pliki źródłowe. Wiązanie zewnętrzne dotyczy nazwy obiektu deklarowanego (nawet wielokrotnie) z użyciem słowa **`extern`** i definiowanego (tylko jednokrotnie) z zasięgiem globalnym bez użycia słowa **`static`**.

- **Nazwa o wiązaniu wewnętrznym** - reprezentuje ten sam obiekt tylko w jednym pliku źródłowym, w innym pliku identyczna nazwa skojarzona jest z innym obiektem.
- **Nazwa bez wiązania** - dotyczy obiektu lokalnego deklarowanego bez słowa **`extern`** lub parametru formalnego funkcji.

Funkcje

W języku C idea podprogramów realizowana jest wyłącznie poprzez definiowanie i wywołanie funkcji. **Każda funkcja musi być przed wywołaniem zadeklarowana**.

Deklaracja funkcji

W deklaracji funkcji podawana jest jej nazwa, typ zwracanej wartości (jeżeli funkcja zwraca wartość), liczba i typ argumentów (jeżeli funkcja wymaga argumentów).

```
extern double pierwiastek(double a, double b, double c);

extern int podzielnik(int k, int n);

extern void rysuj_okrag(float x, float y, float r);

extern float podaj_czas( );
```

Nazwa funkcji w języku C deklarowana bez użycia słowa **`extern`** ma z definicji wiązanie zewnętrzne, czyli obszar widzialności nazwy funkcji obejmuje wszystkie pliki źródłowe składające się na program. Oznacza to, że słowo **`extern`** w deklaracji funkcji jest zbyteczne. Użycie natomiast słowa **`static`** powoduje, że nazwa deklarowanej funkcji uzyskuje wiązanie wewnętrzne. Tak zdefiniowana funkcja może być wywoływana jedynie w obrębie swojego pliku źródłowego. W deklaracji funkcji nie są wymagane nazwy jej argumentów, a jedynie ich typy. Umieszczanie nazw argumentów jest jednak polecane ze względów dokumentacyjnych.

```
double pierwiastek(double,double,double);

int podzielnik(int,int);

void rysuj_okrag(float,float,float);

float podaj_czas( );
```

Definicja funkcji

Funkcja może być wielokrotnie deklarowana, ale w jednym z plików źródłowych musi być umieszczona jej definicja, która ma formę deklaracji uzupełnionej **instrukcją złożoną zawierającą deklaracje i instrukcje ciała funkcji**.

Deklaracja:

```
int nwp(int a, int b);
```

Definicja:

```
int nwp(int a, int b)
{
    int p;
    while(a!=b)
    {
        p= a>b ? a-b : b-a;
        if(a>b) a=p;
            else b=p;
    }
    return p;
}
```

Definicja funkcji może nie występować w pliku źródłowym. W takim przypadku definicja funkcji musi być dostarczona na etapie konsolidacji programu w postaci modułu półskompilowanego (plik z rozszerzeniem .obj) lub biblioteki (plik z rozszerzeniem .lib). Deklaracja funkcji w każdym przypadku jest jednak wymagana.

Uwaga: W bloku funkcji, która zwraca wartość, musi wystąpić instrukcja powrotu **`return`** zawierająca wyrażenie.

Wywołanie funkcji

Wywołanie funkcji dokonywane jest za pośrednictwem **operatora wywołania funkcji ( )**. Podczas wywołania funkcji następuje przekazanie do niej wartości argumentów (jeżeli funkcja ta wymaga argumentów). W języku C wszystkie argumenty przekazywane są **przez wartość**, co oznacza, że w momencie wywołania funkcji przydzielana jest pamięć dla jej argumentów formalnych i kolejno każdemu argumentowi formalnemu przypisywana jest wartość odpowiedniego argumentu aktualnego.

Uwaga: Argumenty funkcji w obszarze jej bloku są obiektami lokalnymi.

Dokonaj analizy poniższego kodu źródłowego funkcji określającej liczbę rzeczywistych pierwiastków równania kwadratowego:

```
int pierwiastek(float a, float b, float c)
{
    float delta;
    delta=b*b-4*a*c;
    if(delta<0)return 0;
        else
            if(delta==0)return 1;
                else return 2;
}
```

Przykład instrukcji zawierających wywołanie ww. funkcji:

```
printf (“Rownanie ma %d rozwiazan”, pierwiastek(x, y, z));
if (pierwiastek(1.5, y, z)==0) continue;
```

Funkcja może być wywoływana **rekurencyjnie**, to znaczy wewnątrz bloku funkcji jest ona ponownie wywoływana.

Rekurencyjne obliczanie silni

Wprowadź i uruchom program, zinterpretuj tekst programu i wyniki:

```
#include <stdio.h>
int silnia(int n){return n?n*silnia(n-1):1;}
int main()
{
  int n;
  printf("Podaj liczbe n=");
  scanf("%d",&n);
  printf("Wartosc silni n!=%d",silnia(n));
  getch();
}
```

```
Podaj liczbe n=6                    KLAWIATURA  6↵
Wartosc silni n!=720
```

Słowo kluczowe *void*

W deklaracjach typów można użyć słowa kluczowego ***void***, które wpisuje się w miejscu nazwy typu. Znaczenie słowa ***void*** w odniesieniu do funkcji:

- Jako typ wyniku funkcji oznacza, że funkcja nie zwraca żadnej wartości.
- Jako treść listy parametrów formalnych funkcji oznacza, że funkcja jest bezparametrowa. Pusta lista parametrów **( )** oznacza również funkcję bez parametrów.

```
void fun(int x);    // funkcja fun nie zwraca wartości

int funk(void);     // funkcja bezparametrowa, równoważnym
                    // zapisem jest  int funk();
```

Struktura modułu źródłowego

Moduł źródłowy jest (nie licząc dyrektyw preprocesora i komentarzy) sekwencją *deklaracji* i *instrukcji* pogrupowanych w *bloki funkcyjne*.

Blok funkcyjny zawiera ujęte w nawiasy klamrowe **{ }** instrukcje wykonywane przy wywołaniu funkcji. Blok funkcyjny może z kolei zawierać *bloki*.

Blok to jedna lub więcej instrukcji zawartych w parze nawiasów **{ }**.

Blok funkcyjny nie może być zawarty w innym bloku lub bloku funkcyjnym. Poza blokiem funkcyjnym mogą występować **jedynie deklaracje**. Każdy blok funkcyjny wraz z poprzedzającym go nagłówkiem stanowi definicję funkcji. **Nagłówek funkcji** ustala nazwę funkcji, liczbę, kolejność i typ argumentów funkcji oraz typ zwracanego przez funkcję rezultatu.

```
int k,n;            // Deklaracja poza blokiem funkcji

int fun1(char z)    // Nagłówek funkcji fun1
{               // Początek bloku funkcyjnego fun1

//... Blok funkcyjny, definicja funkcji fun1

}               // Koniec bloku funkcyjnego, bez średnika

float x;            // Deklaracja poza blokiem funkcji

int fun2(float y)   // Nagłówek funkcji fun2
{               // Początek bloku funkcyjnego fun2
                    // ...
     {          // Początek bloku
                // ...
     }          // Koniec bloku

     {          // Początek bloku
                // ...
          {         // Początek bloku wewnętrznego
                    // ...
          }         // Koniec bloku wewnętrznego
                // ...
     }          // Koniec bloku
                    // ...
}               // Koniec bloku funkcyjnego fun2

int a;              // Deklaracja poza blokiem funkcji
```

Uwaga: W sposób pośredni można wykonać instrukcję poza blokiem funkcyjnym. Do inicjalizacji zmiennych mogą być wykorzystane wyrażenia zawierające funkcje, co pozwala na faktyczne wykonanie dowolnego ciągu instrukcji poza jakąkolwiek funkcją (również poza funkcją **`main`**):

Zasięg nazw obiektów

Zasięg nazwy (identyfikatora) obiektu w programie jest tym obszarem w tekście programu, w którym nazwa ta jest znana i rozpoznawana przez kompilator.

Rodzaje zasięgu identyfikatorów

Zasięg lokalny mają obiekty zdefiniowane w bloku. Zasięg lokalny rozciąga się od miejsca definicji do końca bloku.

Zasięg globalny (zasięg pliku - modułu źródłowego) mają obiekty zdefiniowane na zewnątrz wszelkich bloków (również bloku stanowiącego funkcję). Zasięg globalny rozciąga się od miejsca definicji do końca pliku źródłowego. W innych plikach (ale nie w plikach włączanych dyrektywą *#include*) dany obiekt nie jest znany, chyba że występuje w deklaracji **`extern.`**

```
char x;                 // x ma zasięg globalny
int main()
{
    int y;              // y ma zasięg lokalny (w funkcji)
    {
        char *z;        // z ma zasięg lokalny (w bloku)
    }
}
```

Widzialność i zasłanianie nazw

Obszar widzialności nazwy jest fragmentem programu, w którym można odwoływać się do obiektu danych poprzez tę nazwę. Zmienna globalna może zostać **zasłonięta** przez zmienną lokalną o identycznej nazwie, jak również zmienna lokalna może być zasłonięta przez inną zmienną lokalną o identycznej nazwie w zagnieżdżonym bloku. Oznacza to, że kompilator odwołuje się do *najbardziej zagnieżdżonej* zmiennej lokalnej w jej zasięgu.

Wprowadź i uruchom program, zinterpretuj kod źródłowy:

```
#include <stdio.h>
char x=4;                       // Zmienna globalna
int main()
{
    int x=12;                   // Zmienna lokalna
    printf("\n%d",x);           // Wyświetla liczbę 12
    {
        char x=8;               // Zmienna lokalna zagnieżdżona
        printf("\n%d",x);       // Wyświetla liczbę 8
    }
    printf("\n%d",x);           // Wyświetla liczbę 12
    getch();
}
```

Czas życia obiektu

Obiekt powstaje w momencie, gdy w czasie wykonania programu sterowanie przechodzi przez definicję tego obiektu. **Czas życia obiektu** obejmuje okres w fazie wykonania programu od momentu utworzenia danego obiektu (przydzielenia pamięci dla tego obiektu) do chwili jego usunięcia z pamięci. Pod względem czasu życia obiekty dzielą się na:

- **statyczne**, które trwają przez cały czas wykonania programu,
- **dynamiczne**, które mogą być tworzone i usuwane z pamięci w różnych fazach wykonania programu.

Obiekt statyczny:

- ***Globalny obiekt*** - tworzony tylko jeden raz i żyjący do zakończenia wykonywania programu.
- ***Lokalny obiekt w klasie pamięci static*** - zdefiniowany z użyciem słowa kluczowego **`static`** również żyjący aż do zakończenia wykonywania programu. Inicjalizacja tego obiektu następuje jednorazowo w momencie przechodzenia sterowania przez definicję obiektu.

Obiekt dynamiczny:

- ***Automatyczny obiekt*** - lokalny obiekt zdefiniowany bez podania klasy pamięci **`static`**.
- ***Kontrolowany obiekt*** - w sposób jawny tworzony na stercie i usuwany z pamięci przy pomocy odpowiednich operatorów lub funkcji zarządzających pamięcią.

Automatyczne obiekty lokalne są **wielokrotnie** tworzone i usuwane z pamięci, gdy sterowanie opuści blok, w którym obiekt został zdefiniowany. Inicjalizowanie obiektu lokalnego **powtarzane jest** każdorazowo przy jego tworzeniu.

Uzasadnij wyniki wykonania poniższego programu:

```
#include <stdio.h>
void fun(void)
{
   int k=0;
   static int n=0;
   k=k+1;
   n=n+1;
   printf("\n%d %d",k,n);
}
int main()
{
   fun();
   fun();
   fun();
}
```

```
1 1

1 2

1 3
```

Uwaga: W definicji obiektu lokalnego może występować opcjonalnie słowo kluczowe **`auto`**, które określa obiekt *automatyczny*.

Równoważne definicje:

```
auto int x;
```

```
int x;
```

Wiązanie nazwy obiektu

(powtórzenie i uzupełnienie)

Po kompilacji programu z każdą nazwą obiektu musi być związany jednoznacznie obiekt, który ta nazwa oznacza. Pewna nazwa może występować w wielu różnych miejscach pliku źródłowego, a nawet w różnych plikach źródłowych. Rodzaj **wiązania nazwy** określa sposób, w jaki danej nazwie przypisywany jest obiekt.

Rodzaje wiązań:

- **Nazwa o wiązaniu zewnętrznym** - we wszystkich plikach źródłowych tworzących program oznacza ten sam obiekt. Tylko w jednym miejscu występuje definicja obiektu, w pozostałych jedynie deklaracje. Obszar widzialności takiej nazwy rozciąga się na inne pliki źródłowe. Wiązanie zewnętrzne dotyczy nazwy obiektu deklarowanego (nawet wielokrotnie) z użyciem słowa **`extern`** i definiowanego (tylko jednokrotnie) z zasięgiem globalnym bez użycia słowa **`static`**.

- **Nazwa o wiązaniu wewnętrznym** - reprezentuje ten sam obiekt tylko w jednym pliku źródłowym, w innym pliku identyczna nazwa skojarzona jest z innym obiektem. Wiązanie wewnętrzne mają nazwy obiektów globalnych definiowane ze słowem **`static`**. Wiązanie wewnętrzne dotyczy również funkcji definiowanej ze słowem **`static`**. Tak zdefiniowana funkcja może być wywoływana jedynie w obrębie swojego pliku źródłowego.

- **Nazwa bez wiązania** - dotyczy obiektu lokalnego deklarowanego bez słowa **`extern`** lub parametru formalnego funkcji.

Wartość początkowa zmiennych

W przypadku, gdy zmienna nie jest inicjalizowana w sposób jawny, otrzymuje wartość początkową **zero**. Wyjątek stanowią zmienne **automatyczne**, które otrzymują **nieokreślone** wartości początkowe.

Dokonaj analizy programu, zwróć uwagę na komentarze:

```
#include <stdio.h>
char k,y[4];                        // Inicjalizacja zerem
int main()
{
    printf("\n(%d) %d %d %d %d",k,y[0],y[1],y[2],y[3]);

    char k,y[4];                    // Brak inicjalizacji
    printf("\n(%d) %d %d %d %d",k,y[0],y[1],y[2],y[3]);

    char i=5,x[4]={11,12,13};       // Inicjalizacja
    printf("\n(%d) %d %d %d %d",i,x[0],x[1],x[2],x[3]);
    {
        char i,x[4];                // Brak inicjalizacji
        printf("\n(%d) %d %d %d %d",i,x[0],x[1],x[2],x[3]);
    }
    {
        char i=3,x[4]={7,13};       // Inicjalizacja
        printf("\n(%d) %d %d %d %d",i,x[0],x[1],x[2],x[3]);
    }
    getch(); return 0;
}
```

```
(0) 0 0 0 0
(141) 184 142 194 2
(5) 11 12 13 0
(50) 184 142 184 142
(3) 7 13 0 0
```

Zadanie C08-1

a) Zdefiniuj 1000-elementową tablicę znaków **tab** jako obiekt globalny.

b) Napisz **funkcję** `tekst` (**typu** `void`) wczytującą do tablicy **tab** (i jednocześnie drukującą) dowolny tekst z klawiatury do chwili wciśnięcia „Enter” (*zakładamy bez sprawdzania, że długość tekstu nie przekracza rozmiaru tablicy* ***tab***); **wskazówka:** skorzystaj z funkcji `scanf("%s",tab)` – porównaj zajęcia C02.

c) Napisz **funkcję** `znak` (**typu** `int`) przeszukującą tablicę **tab** oraz zwracającą liczbę wystąpień małej litery **a** we wczytanym do niej tekście. *Uwaga: przeszukiwanie tablicy (tekstu) należy zakończyć z chwilą napotkania znaku „Enter”.*
Wskazówki:
- użyj pętli do – while,
- koniec łańcucha (Enter) rozpoznawany jest jako znak o kodzie 10.

d) Napisz **funkcję** `main`, która wywołuje funkcję `tekst`, a następnie funkcję `znak`, po czym drukuje wartość zwróconą przez tą funkcję. Uruchom program.

Kod źródłowy programu C08-1 wraz z oknem zawierającym efekty przykładowego uruchomienia tego programu, należy zapisać w **jednym** dokumencie (**format Word**) o nazwie **C08.doc** i przesłać za pośrednictwem platformy Moodle w wyznaczonym terminie.

Zadanie dodatkowe nadobowiązkowe – dla zainteresowanych – możliwość oddania wyłącznie w postaci papierowego wydruku najpóźniej na następnych zajęciach, wraz z jednoczesnym ustnym zaliczeniem zadania (wyjaśnienie tekstu źródłowego).

Napisz **funkcję** `znaki` (**typu int**) wczytującą dowolny tekst (max. 990 znaków), drukującą wczytywany tekst, oraz następnie drukującą tabelę częstości występowania znaków zawartych w tekście (rozważamy 256 znaków o kodach 0-255, przy czym częstość ma być drukowana dla znaków występujących co najmniej jeden raz, bez znaku LF wprowadzonego Enter-em). Funkcja zwraca całkowitą liczbę znaków w tekście.

Napisz funkcję `main`, która wywołuje funkcję `znaki`, po czym drukuje całkowita liczbę znaków we wprowadzonym tekście. Uruchom program. Poniżej wydruku tekstu źródłowego umieść okno z efektami przykładowego uruchomienia programu.