

Wojskowa Akademia Techniczna  
Wydział Elektroniki  
Instytut Telekomunikacji

Studia I°

# Języki C/C++ w zastosowaniach sieciowych

Materiały pomocnicze do zajęć  
Dodatek 2

dr inż. Jarosław Krygier

Warszawa 2020

## Spis treści

1	Listy wiązane.....	3
1.1	Plik główny.....	3
1.2	Plik nagłówkowy.....	5
1.3	Plik funkcji.....	5
2	Biblioteki dynamiczne.....	7
2.1	Plik: biblioteki_dynamiczne.c.....	7
2.2	Plik: biblioteka.c.....	8
3	Wskaźniki do funkcji.....	9

# 1 Listy wiązane

## 1.1 Plik główny

```
/*
=====
Name      : wyklad.c
Author    : Krygier
Version   :
Copyright : WEL WAT
Description : Języki C/C++ W zastosowaniach sieciowych - Przykładowy program do testowania list
wiaznych
=====
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h> //dla deklaracji memset i memcpy
#include "plik_naglowkowy.h"

#define MAX_BUFF_SIZE 20; //20 pakietow

// deklaracje funkcji
int dodaj_element ();
int usun_pierwszy_element ();

int main(void) {
    /// @brief Funkcja main
    /**
     * @param void Brak atrybutow
     * @return Program zwraca \a int
     */
    int i, n, l; // zmienne pomocnicze
    struct bufor *adres_zakotwiczenia = NULL;
    struct bufor *obceny_adres = NULL;
    struct bufor *ostatni = NULL;
    struct bufor *temp;
    unsigned char dane[1500];
    int max_bufor = MAX_BUFF_SIZE;
    memset (dane, 0, 1500); //wyzerowanie tablicy

    for (i=0;i<10;i++){
        //pobierz dane
        memset (dane, i+1, 1500); //nowe dane
        //sprawdz czy bufor (lista) jest pusta
        if (adres_zakotwiczenia == NULL) {
            //adres zakotwiczenia jest NULL -> lista jest pusta
            //przydziel dynamicznie pamiec dla elementu listy
            adres_zakotwiczenia = (struct bufor*) malloc (sizeof(struct bufor));
            obceny_adres = adres_zakotwiczenia;
            obceny_adres->pierwszy = adres_zakotwiczenia;
            obceny_adres->poprzedni = NULL;
            obceny_adres->nastepny = NULL;
            memcpy (obceny_adres->pakiet, dane, 1500);
            obceny_adres = NULL;
        } else {
            //adres zakotwiczenia nie jest NULL, lista nie jest pusta
            //znajdz ostatni element listy
            ostatni = adres_zakotwiczenia;
            for(n=0;n<max_bufor;n++){
                if (ostatni->nastepny != NULL) {
                    ostatni = ostatni->nastepny;
                }
            }
        }
    }
}
```

```

        }
        else {
            break;
        }
    }
    obceny_adres = (struct bufor*) malloc (sizeof(struct bufor));
    ostatni->nastepny = obceny_adres;
    obceny_adres->pierwszy = adres_zakotwiczenia;
    obceny_adres->poprzedni = ostatni;
    obceny_adres->nastepny = NULL;
    memcpy (obceny_adres->pakiet, dane, 1500);
}
}

//zliczaj liczbe elementow
l=0;
for(temp = adres_zakotwiczenia; temp != NULL; temp=temp->nastepny){
    l++;
}
printf ("liczba elementow: %d\n", l);

memset (dane, 0xaa, 1500); //nowe dane
dodaj_element (adres_zakotwiczenia, dane);

//zliczaj liczbe elementow
l=0;
for(temp = adres_zakotwiczenia; temp != NULL; temp=temp->nastepny){
    l++;
}

printf ("liczba elementow: %d\n", l);

memset (dane, 0x00, 1500);

usun_pierwszy_element (&adres_zakotwiczenia, dane);

//zliczaj liczbe elementow
l=0;
for(temp = adres_zakotwiczenia; temp != NULL; temp=temp->nastepny){
    l++;
}

printf ("liczba elementow: %d\n", l);

puts("Koniec");
return EXIT_SUCCESS;
}

```

## 1.2 Plik nagłówkowy

```
/*
 * plik_naglowkowy.h
 * Author: Krygier
 */

#ifndef PLIK_NAGLOWKOWY_H_
#define PLIK_NAGLOWKOWY_H_

struct bufor {
    struct bufor *nastepny; ///< wskaznik na nastepny element listy
    struct bufor *poprzedni; ///< wskaznik na poprzedni element listy
    struct bufor *pierwszy; ///< wskaznik na pierwszy element listy
    unsigned char pakiet[1500];
};

#endif /* PLIK_NAGLOWKOWY_H_ */
```

## 1.3 Plik funkcji

```
/*
 * funkcje.c
 * Author: Krygier
 */

#include <stdio.h>
#include <stdlib.h>
#include "plik_naglowkowy.h"

int dodaj_element (struct bufor *adres_zakotwiczenia, char *dane) {

    int n;
    struct bufor *obceny_adres = NULL;
    struct bufor *ostatni = NULL;
    //unsigned char dane[1500];

    //memset (dane, 0xaa, 1500);
    //sprawdz czy bufor (lista) jest pusta
    if (adres_zakotwiczenia == NULL) {
        //adres zakotwiczenia jest NULL -> lista jest pusta
        //przydziel dynamicznie pamiec dla elementu listy
        adres_zakotwiczenia = (struct bufor*) malloc (sizeof(struct bufor));
        obceny_adres = adres_zakotwiczenia;
        obceny_adres->pierwszy = adres_zakotwiczenia;
        obceny_adres->poprzedni = NULL;
        obceny_adres->nastepny = NULL;
        memcpy (obceny_adres->pakiet, dane, 1500);
        obceny_adres = NULL;
    } else {
        //adres zakotwiczenia nie jest NULL lista nie jest pusta
        //znajdz ostatni element listy
        ostatni = adres_zakotwiczenia;
        for(n=0;n<20;n++){
            if (ostatni->nastepny != NULL) {
                ostatni = ostatni->nastepny;
            }
            else {
                break;
            }
        }
        obceny_adres = (struct bufor*) malloc (sizeof(struct bufor));
        ostatni->nastepny = obceny_adres;
        obceny_adres->pierwszy = adres_zakotwiczenia;
    }
}
```

```

        obceny_adres->poprzedni = ostatni;
        obceny_adres->nastepny = NULL;
        memcpy (obceny_adres->pakiet, dane, 1500);
    }

    return (1);
}

int usun_pierwszy_element (struct bufor **adres_zakotwiczenia, char *dane) {

    struct bufor *temp;
    ///Funkcja do usuwania elementu listy, zwraca wskaznik do skopiowanych danych
    if (*adres_zakotwiczenia !=NULL) {
        //sprawdz czy jest nastepny element
        if ((*adres_zakotwiczenia)->nastepny != NULL) {
            (*adres_zakotwiczenia)->nastepny->pierwszy=(*adres_zakotwiczenia)->nastepny;
            (*adres_zakotwiczenia)->nastepny->poprzedni = NULL;
            temp = (*adres_zakotwiczenia)->nastepny;

            memcpy (dane, (*adres_zakotwiczenia)->pakiet, 1500);

            free (*adres_zakotwiczenia);

            *adres_zakotwiczenia = temp;

        }
        else {
            memcpy (dane, (*adres_zakotwiczenia)->pakiet, 1500);

            free (*adres_zakotwiczenia);

            *adres_zakotwiczenia = NULL;
        }
    }
    else {
        //nie mam nic do skasowania
        return (-1);
    }
    return (1);
}

```

## 2 Biblioteki dynamiczne

### 2.1 Plik: biblioteki\_dynamiczne.c

```
/*
=====
Name       : biblioteki_dynamiczne.c
Author      : Krygier
Version     : 1.0
Copyright   : Tylko na potrzeby zajęć
Description : Języki C/C++ W zastosowaniach sieciowych - Biblioteki dynamiczne
=====
*/
//Utworzenie biblioteki dynamicznej:
//gcc -shared biblioteka.c -o biblioteka.so

/*
Aby mieć 'dynamic library' (dl), linker musi łączyć z opcja -l: dl
Ustawić w Eclipse: C/C++ -> Settings ->GCC C Linker -> Libraries
Program może ładować w trakcie pracy potrzebne mu biblioteki. Linker dostarcza w tym celu 4
funkcji:
    dlopen ładowanie biblioteki,
    dlsym zwrócenie wskaźnika do odpowiedniego symbolu (funkcji) w bibliotece,
    dlerror obsługa błędów,
    dlclose zamykanie biblioteki.
Funkcje te są udostępniane poprzez nagłówek dlfcn.h (trzeba go dodać do głównego programu).

#include <dlfcn.h>
void *dlopen(const char *filename, int flag);
char *dlerror(void);
void *dlsym(void *handle, const char *symbol);
int dlclose(void *handle);
*/

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

int main(void) {
    puts("Dolaczanie bibliotek dynamicznych");
    puts("=====");

    int status_wyjscia; // do dlclose

    void *Biblioteka; // wskaźnik do biblioteki
    int (*Funkcja)(int, int); //wskaźnik do funkcji
    double (*Funkcja1)(int, int); //wskaźnik do funkcji1

    Biblioteka = dlopen("./biblioteka.so", RTLD_LAZY); // otwarcie biblioteki RTLD_LAZY - wskaźniki
na funkcje wewnętrzne są budowane dopiero w trakcie wywołania dlsym
    if(!Biblioteka) {
        printf ("Error otwarcia: %s\n", dlerror());
        return(1);
    }
    Funkcja = dlsym(Biblioteka, "suma"); // pobranie wskaźnika na odpowiednia funkcje podana za
pomocą nazwy

    printf ("suma:%d \n", Funkcja (4, 2));

    Funkcja = dlsym(Biblioteka, "iloczyn");

    printf ("iloczyn:%d \n", Funkcja (4, 2));

    Funkcja = dlsym(Biblioteka, "roznica");
```

```

    if (Funkcja== NULL) {
        printf ("Bład wywołania funkcji 'roznica()': brak w bibliotece\n");
    } else {
        printf ("roznica:%d \n", Funkcja (8, 2));
    }

    status_wyjścia = dlclose (Biblioteka); //zamknięcie biblioteki - > jeśli poprawnie to
status_wyjścia = 0
    if(status_wyjścia) { // jeśli 0 (fałsz) -> dlopen() zamknęła bibliotekę poprawnie
        printf ("Error zamknięcia: %s\n", dlerror());
        return(1);
    }

    //=====
    Biblioteka = dlopen("./biblioteka.so", RTLD_NOW); //RTLD_NOW - wskaźniki na funkcje wewnętrzne
sa tu budowane potem dlsym je wykorzystuje
    if(!Biblioteka) {
        printf ("Error otwarcia: %s\n", dlerror());
        return(1);
    }
    Funkcja1 = (double (*)(int, int)) dlsym(Biblioteka, "iloraz"); //z podpowiedzią jaki wskaźnik
do jakiej funkcji będzie zwracany

    printf ("iloraz:%f \n", Funkcja1 (5, 3));

    status_wyjścia = dlclose (Biblioteka);
    if(status_wyjścia) { // jeśli 0 (fałsz) -> dlopen() zamknęła bibliotekę poprawnie
        printf ("Error zamknięcia: %s\n", dlerror());
        return(1);
    }

    return EXIT_SUCCESS;
}

```

## 2.2 Plik: biblioteka.c

```

/*
 * biblioteka.c
 * Author: Jarosław Krygier
 * Tylko na potrzeby zajęć
 */

//Utworzenie biblioteki dynamicznej:
//gcc -shared biblioteka.c -o biblioteka.so

#include <stdio.h>

int suma (int a, int b){
    return (a+b);
}

int iloczyn (int a, int b){
    return (a*b);
}

double iloraz (int a, int b){
    return ((double)a / (double) b);
}

/*int roznica (int a, int b){
    return (a-b);
}*/

```



### 3 Wskaźniki do funkcji

```
/*
=====
Name       : W_PRC2.c
Author      : Krygier
Version     : v1
Copyright   : Your copyright notice
Description : Języki C/C++ W zastosowaniach sieciowych, Wskazniki do funkcji
=====
*/

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

//funkcja dodawania
int dodawanie(int x, int y) {
    return (x+y);
}

//funkcja odejmowania
int odejmowanie (int x, int y) {
    return (x-y);
}

int operacja (int (*) (int, int), int, int); // definicja ponizej funkcji main()

int main(void) {
    puts("Wyklad1"); /* Wyklad1 PRC2*/
    //Zaawansowane operacje na wskaznikach

    int a;
    int b=5;
    int c=6;
    int* wsk;

    ////////////////
    //wskazniki
    ////////////////

    wsk = &a;
    printf ("Adres zmiennej 'a': 0x%x\n", (unsigned int) wsk); // rzutowanie na unsigned int,
    //poniewaz printf wymaga takiego typu (nie ma warningow)
    wsk = &b;
    printf ("Adres zmiennej 'b': 0x%x\n", (unsigned int) wsk);
    wsk = &c;
    printf ("Adres zmiennej 'c': 0x%x\n", (unsigned int) wsk);

    ////////////////
    //wskazniki do funkcji
    ////////////////
    //deklaracja
    int (*funkcja) (int, int);
    //jaka jest roznica z int *funkcja (int, int);

    funkcja = &dodawanie;
    // teraz wskaznik wskazuje na funkcje 'dodawanie'
    printf ("1) Wskaznik na funkcje 'dodawanie': 0x%x\n", (unsigned int) funkcja);

    //lub alternatywnie
    funkcja = dodawanie;
    // teraz wskaznik wskazuje na funkcje 'dodawanie'
    printf ("2) Wskaznik na funkcje 'dodawanie': 0x%x\n", (unsigned int) funkcja);
}
```

```

//Wywołanie funkcji przez wskanik
a = funkcja (b,c);
printf ("1) Wynik: %d \n", a);
// lub alternatywnie
a = (*funkcja) (b,c);
printf ("2) Wynik: %d \n", a);

// przypisanie do wskaźnika różnych funkcji
//teraz ten sam wskaźnik wskazuje na inną funkcję
funkcja = odejmowanie; //można przypisać funkcję bez &
printf ("3) Wskaźnik na funkcję 'odejmowanie': 0x%x\n", (unsigned int) funkcja);
//wywołanie jest identyczne, ale wynik różny
a = (*funkcja) (b,c);
printf ("3) Wynik: %d \n", a);

//////////
//tablica wskaźników do funkcji
//////////

// deklaracja tablicy wskaźników do funkcji
int (*tablica_wskaźników_do_funkcji [3]) (int, int);

//przypisanie adresów funkcji do komórek tablicy
tablica_wskaźników_do_funkcji [0] = dodawanie;
tablica_wskaźników_do_funkcji [1] = odejmowanie;
tablica_wskaźników_do_funkcji [2] = NULL;

printf ("Wskaźnik na funkcję 'dodawanie' [0]: 0x%x\n", (unsigned int)
tablica_wskaźników_do_funkcji [0]);
printf ("Wskaźnik na funkcję 'odejmowanie'[1]: 0x%x\n", (unsigned int)
tablica_wskaźników_do_funkcji [1]);
printf ("Wskaźnik na funkcję 'NULL'[2]: 0x%x\n", (unsigned int) tablica_wskaźników_do_funkcji
[2]);

//wywołanie funkcji z tablicy

a = (*tablica_wskaźników_do_funkcji [0]) (b,c);
printf ("1) Wynik: %d \n", a);
a = (*tablica_wskaźników_do_funkcji [1]) (b,c);
printf ("2) Wynik: %d \n", a);

//lub alternatywnie
a = tablica_wskaźników_do_funkcji [0] (b,c);
printf ("1) Wynik: %d \n", a);
a = tablica_wskaźników_do_funkcji [1] (b,c);
printf ("2) Wynik: %d \n", a);

/*UWAGA!!!*/
/*
 * Wskaźniki do funkcji można:
 - przekazywać do innych funkcji jako argumenty
 - zwracać jako wynik działania funkcji
 - porównywać z NULL
 * Nie można
 - wykonywać operacji matematycznych
 */

//Przekazanie wskaźnika do funkcji jako argumentu innej funkcji
funkcja = dodawanie;
a = operacja (funkcja, 3, 5);
printf ("1) Wynik: %d \n", a);
//lub bezpośrednio nazwa funkcji jest też wskaźnikiem
a = operacja (dodawanie, 3, 5);
printf ("2) Wynik: %d \n", a);

// a teraz przekazujemy wskaźnik innej funkcji

```

```

a = operacja (odejmowanie, 3, 5);
printf ("3) Wynik: %d \n", a);

//sprawdzenie
//Napisac fukcje 'operacja_arytm', ktora wykonuje y=(a/b)+c
//Napisac fukcje o nazwie 'operacje_new' z argumentami (typ_operacji, arg1, arg2, arg3)
zwracajaca wynik:
// dodawania lub odejmowania, lub wynik fukcji 'operacja_arytm' w zaleznosci od przekazanego
typu operacji

return EXIT_SUCCESS;
}

// fukcja z przekazany wskaznikiem do fukcji
int operacja (int (*wsk_do_funkcji_przekazany) (int, int), int a, int b){ // jako pierwsza bedzie
przekazywany wskaznik do fukcji
int wynik;

wynik = wsk_do_funkcji_przekazany (a,b);
// lub
//wynik = (*wsk_do_funkcji_przekazany) (a,b);

return wynik;
}

```