Laboratorium Informatyki
Programowanie w języku C
Ćwiczenie 3

1 Wstęp

Specyficznym typem zmiennych stosowanym w języku C i C++ jest wskaźnik. Jest to zmienna przechowująca informacje o adresie w pamięci pod którym zmienna na którą wskazuje przechowuje soja wartość. Wskaźniki znajdują szerokie zastosowanie w kodzie pisanym w języku C.

Standardowy typ zmiennej pozwala na przechowywanie w pamięci pojedynczej wartości danego typu. W wielu sytuacjach istnieje konieczność przechowywania większej liczby danych. Deklarowanie dla każdej z nich osobnej zmiennej było by nieefektywne. W tego typu sytuacjach stosuje się tablice.

Kod programu w języku C może zostać podzielony na kilka plików. Zazwyczaj wydziela się dla każdego pliku źródłowego (*.c) plik nagłówkowy (*.h) w którym umieszcza się kod z części nagłówkowej, podpięcie bibliotek, deklaracje funkcji i zmiennych globalnych. Dzięki temu można uzyskać większa czytelność kodu. W pliku źródłowym podpina się plik nagłówkowy dyrektywą `#include "nazwa.h"`.

2 Wskaźniki

Wskaźnik jest specyficznym typem zmiennej funkcjonującej w połączeniu z inną zmienną. Przechowuje on informacje o adresie w pamięci w którym wskazywana zmienna przechowuje swoją zawartość. Wskaźnik może wskazywać tylko na zmienną tego samego typu co on. Deklaracja wskaźnika jest podobna do deklaracji zmiennej, tylko nazwa wskaźnika poprzedzona jest znakiem `*`. Aby przypisać do wskaźnika adres zmiennej, należy skorzystać z operatora referencji `&`. Odwołując się bezpośrednio do wskaźnika zostanie zwrócony adres na który wskazuje. Jeżeli chcemy odwołać się do wskazywanej wartości należy odwołać się do wskaźnika poprzedzając jego nazwę znakiem `*`. Podobnie postępuje się chcąc przypisać wartość do wskazywanej przez wskaźnik zmiennej.

```
#include <stdio.h>
int main()
{
   int x = 55;
   int *w;
   w = &x;
   printf("Zmienna x=%i, wskaznik w=0x%x, wskazuje *w=%i\n",x,w,*w);
   *w = 110;
   printf("Zmienna x=%i, wskaznik w=0x%x, wskazuje *w=%i\n",x,w,*w);
}
```

2.1 Wskaźnik a funkcja

W podstawowej konstrukcji funkcji gdzie parametrami wejściowymi są zmienne, przekazując do funkcji parametr tworzona jest na czas jej działania kopia zmiennej. Jeżeli w trakcie działania funkcji wartość takiej zmiennej ulegnie zmianie, nie wpłynie to w żaden sposób na wartość przekazywanego parametru.

```
#include <stdio.h>
void test(int x) {
   x++;
   printf("x=%i\n",x);
}
int main()
{
   int x = 1;
   printf("x=%i\n",x);
   test(x);
   printf("x=%i\n",x);
}
```

Inaczej sytuacja wygląda gdy jako parametr wejściowy funkcji zastosujemy wskaźnik. Wtedy do funkcji przekazywany jest adres komórki w pamięci w którym przekazywana do funkcji zmienna przechowuje swoją zawartość. W takim przypadku funkcja działająca na wskaźnikach zmieniając wartość zmiennej zmieni zawartość komórki pamięci, a więc w konsekwencji wartość zmiennej która przekazuje swoją zawartość do funkcji.

```
#include <stdio.h>
void test(int *x) {
   (*x)++;
   printf("x=%i\n",*x);
}
```

```
int main()
{
   int x = 1;
   printf("x=%i\n",x);
   test(&x);
   printf("x=%i\n",x);
}
```

Dzięki takiemu rozwiązaniu można zapisać funkcję która będzie zwracała dwie, lub więcej wartości, w zależności od liczby parametrów wejściowych funkcji typu wskaźnikowego.

3 Tablice

Tablica pozwala na przechowywanie pod jedną nazwą wielu wartości danego typu. Poszczególne elementy tablicy są indeksowane od `0` do `n-1`, gdzie `n` jest liczbą elementów danej tablicy. Deklarując tablicę określa się typ elementu tablicy, nazwę tablicy oraz liczbę elementów jaką dana tablica będzie przechowywać.

```
typ nazwa[rozmiar]; //składnia
int liczby[20];
char nazwa[20];
float kurs[10];
```

Deklarując tablice kompilator rezerwuje w pamięci rozmiar potrzebny do przechowywania n wartości danego typu. Nazwa tablicy jest wskaźnikiem na jej pierwszy element. Odwołując się do danego elementu tablicy należy podać jego nazwę i numer indeksu. Drugim sposobem jest odwoływanie się za pomocą wskaźnika.

```
#include <stdio.h>
int main()
{
    int tab[2];
    tab[0]=5;
    *(tab+1)=6;
    printf("tab[0]=%i, tab[1]=%i",*tab,tab[1]);
}
```

Odwołanie do poszczególnych elementów tablicy wygodnie jest realizować za pomocą pętli `for()`, odwołując się do poszczególnych elementów tablicy poprzez parametr iteracyjny pętli.

Po zadeklarowaniu elementy tablicy będą przechowywały losowe wartości. Podobnie jak w przypadku zmiennych, tablicę można także inicjować ustawiając wartości początkowe dla poszczególnych elementów.

Inicjacja tablicy może być przeprowadzona na kilka sposobów, jak pokazano poniżej. Można dla tablicy o zadeklarowanym rozmiarze przypisać wymaganą liczbę elementów. Przypisując do tablicy określoną liczbę elementów nie trzeba podawać jej rozmiaru, zostanie on dobrany automatycznie, tak aby pomieścić wszystkie elementy z listy. Jeżeli zostanie przypisana mniejsza liczba elementów niż podany rozmiar tablicy, do nieokreślonych elementów zostanie przypisana wartość `0`. Można także przypisać wartości do konkretnych elementów tablicy.

```
#include <stdio.h>
int main()
{
   int tab1[5] = {1,2,3,4,5};
   int tab2[] = {10,11,12,13,14};
   int tab3[5] = {0};
   int tab4[] = {[0]=5,[3]=9};
   int tab5[5] = {[0]=2,[2]=5};
   printf("tab1\ttab2\ttab3\ttab4\ttab5\n");
   for(int i=0;i<5;i++)
   printf("%4i\t%4i\t%4i\t%4i\t%4i\n",
      tab1[i],tab2[i],tab3[i],tab4[i],tab5[i]);
}
```

Przy deklaracji lub inicjacji tablicy musi zostać określony jej rozmiar. Jest to niezbędne do zarezerwowania wymaganego obszaru w pamięci. Rozmiar tablicy można określić wykorzystując funkcję `sizeof()`. Funkcja ta zwraca liczbę bajtów zajętą przez tablicę. Chcąc uzyskać informację o liczbie elementów, należy zwróconą przez funkcję wartość podzielić przez rozmiar zajmowany przez pojedynczy element tablicy.

```
#include <stdio.h>
int main()
{
   short int tab1[5] = {1,2,3,4,5};
   double tab2[5] = {1,2,3,4,5};
   printf("Tablica tab1 zajmuje w pamieci %i bitow
   i zawiera %i elementow\n",
      sizeof(tab1),sizeof(tab1)/sizeof(short int));
   printf("Tablica tab2 zajmuje w pamieci %i bitow
   i zawiera %i elementow\n",
      sizeof(tab2),sizeof(tab2)/sizeof(*tab2));
}
```

3.1 Tablice wielowymiarowe

W języku C można deklarować także tablice wielowymiarowe. Rzadko jednak korzysta się z większych tablic jak tablice dwuwymiarowe. Deklamując tablicę dwuwy-

miarową należy w określić liczbę wierszy i liczbę elementów w wierszu. Przy deklaracji tablicy dwuwymiarowej obowiązują te same zasady co w przypadku tablicy jednowymiarowej.

```
#include <stdio.h>
int main()
{
   int tab1[3][3];
   int tab2[3][3] = {{1,2,3},{3,2,1},{2,3,1}};
   int tab3[3][3] = {4,4,4,5,5,5,6,6,6};
   int tab4[3][3] = {0};
   int tab5[3][3] = {[0][0]=4,[2][2]=5};
   for(int i=0;i<3;i++) {
      for(int j=0;j<3;j++)
         printf("%i ", tab2[i][j]);
      printf("\n");
   }
}
```

Inicjując tablicę danymi można nie podawać informacji o liczbie elementów w wierszu. Warunkiem jest przypisanie danych w taki sposób aby kompilator mógł określić rozmiar tablicy.

```
   int tab2[][3] = {{1,2,3},{3,2,1},{2,3,1}};
   int tab3[][3] = {4,4,4,5,5,5,6,6,6};
   int tab5[][3] = {[0][0]=4,[2][2]=5};
```

3.2 Tablice a funkcje

Tablica może być parametrem wejściowym funkcji, funkcja nie może jednak zwracać zmiennej tablicowej. Cechą charakterystyczną zmiennej tablicowej jest to że jest ona rozbudowaną formą wskaźnika. Powoduje to, że w sytuacji gdy jest ona parametrem wejściowym funkcji, przekazywana jest jako wskaźnik. Co w efekcie sprawia że jeżeli w wyniku działania funkcji ulegnie zmianie jej zawartość to zmieni się także zawartość tablicy przekazanej w wywołaniu funkcji.

```
#include <stdio.h>
void multi(int x[3][3])
{
   for(int i=0;i<3;i++)
      for(int j=0;j<3;j++)
         x[i][j] = (i+1)*(j+1)*x[i][j];
}
```

```
int main()
{
   int tab[3][3] = {{1,2,3},{3,2,1},{2,3,1}};
   multi(tab);
   for(int i=0;i<3;i++) {
      for(int j=0;j<3;j++)
         printf("%i ", tab[i][j]);
      printf("\n");
   }
}
```

Deklarując funkcję i tablica jako parametrem wejściowym należy:

- określić dokładnie jej rozmiar,
- podać przynajmniej liczbę kolumn,
- skorzystać z wskaźnika,
- przekazać rozmiar jako parametry.

W przypadku skorzystania z wskaźnikowego przekazywania parametry, trzeba przekazać także informacje o rozmiarze tablicy. Funkcja `sizeof()` nie pozwoli na uzyskanie informacji o rozmiarze tablicy.

```
#include <stdio.h>
void multi(int size,int *x)
{
   printf("Rozmiar tablicy w funkcji: %i\n", sizeof(x)/sizeof(int));
   for(int i=0;i<size;i++)
   x[i] = (i+1)*x[i];
}
int main()
{
   int tab[3] = {1,2,3};
   int s = sizeof(tab)/sizeof(int);
   printf("Rozmiar tablicy: %i\n",s);
   multi(s,tab);
   for(int i=0;i<3;i++)
   printf("%i ", tab[i]);
}
```

Za pomocą zmiennej wskaźnikowej można przekazać statycznie deklarowaną tablicę tylko pierwszego rzędu. Przekazywanie tablic wyższych rzędów też jest możliwe, ale muszą zostać zadeklarowane dynamicznie.

3.3 Tablice dynamiczne

W wielu sytuacjach pisząc kod programu nie jesteśmy wstanie z góry określić rozmiaru tablicy jaki będzie konieczny do wykonania. Czasami zachodzi też sytuacja zmiany rozmiaru tablicy w trakcie działania programu. Powoduje to że przy korzystaniu ze statycznych tablic konieczne by było deklarowanie tablicy o nadmiarowym rozmiarze. Co jest nieefektywne pod kontem wykorzystania pamięci. W niektórych, prostych sytuacjach można korzystać z deklaracji parametrycznej tablicy, gdzie rozmiar tablicy określany jest przez zmienne. Tak deklarowanych tablic się nie inicjuje.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int main()
{
   int m,n;
   srand(time(NULL));
   printf("Podaj rozmiar tablicy (m,n): ");
   scanf("%i,%i",&m,&n);
   int tab[m][n];
   for(int i=0;i<m;i++)
      for(int j=0;j<n;j++)
         tab[i][j] = rand()%100;
   for(int i=0;i<m;i++){
      for(int j=0;j<n;j++)
         printf("%2i ", tab[i][j]);
      printf("\n");
   }
}
```

Znacznie efektywniejszym rozwiązaniem jest dynamiczne przydzielanie pamięci dla zmiennej. Tablice dynamiczne deklaruje się w oparciu o zadeklarowane zmienne wskaźnikowe odpowiedniego stopnia. Dla tablicy dwuwymiarowej będzie to zmienna o podwójny wskaźniku. Dla zadeklarowanego wskaźnika można przydzielić wymagany obszar pamięci konieczny do przechowania określonej liczby elementów tablicy. Zadanie to można zrealizować jedną z dwóch funkcji `malloc()` lub `calloc()`.

```
zmienna = (typ*)malloc(liczba*sizeof(zmienna));
zmienna = (typ*)calloc(liczba,sizeof(zmienna));
```

Obydwie funkcje rezerwują dla wskazanej zmiennej obszar pamięci w bajtach określany przez wielokrotność pamięci wymaganej przez jedną komórkę tablicy. Funkcja `calloc()` dodatkowo ustawia wartość każdej komórki tablicy na `0`.

W odróżnieniu od klasycznie zadeklarowanych zmiennych lokalnych które po zakończeniu działania funkcji (bloku kodu) są usuwane z pamięci, pamięć zarezerwowana

dla zmiennych alokowanych pozostaje zablokowywana. Koniecznie jest ręczne zwolnienie pamięci przydzielanej danej zmiennej funkcją `free()`.

Zmienne dynamiczne można w kodzie relokować, zmieniając ich rozmiar. Za pomocą funkcji `realloc()` można zmienić rozmiar tablicy (zwiększyć lub zmniejszyć) lub skopiować dane z jednej tablicy do drugiej. Funkcja `realloc()` działa podobnie jak `malloc()` modyfikując tylko rozmiar zarezerwowanego obszaru.

```
#include <stdio.h>
#include <stdlib.h>
void drukuj(int,int*);
int main()
{
   int *tab1, *tab2;
   printf("Podaj rozmiar tablicy: ");
   int n;
   scanf("%i",&n);
   tab1 = (int*)malloc(n*sizeof(tab1[0]));
   printf("Tablica 1:\n");
   drukuj(n,tab1);
   tab2 = (int*)calloc(n,sizeof(tab2[0]));
   printf("Tablica 2:\n");
   drukuj(n,tab2);
   tab1 = realloc(tab1,2*n*sizeof(tab1));
   printf("Tablica 1:\n");
   drukuj(2*n,tab1);
   tab2 = realloc(tab2,2*n*sizeof(tab2));
   printf("Tablica 2:\n");
   drukuj(2*n,tab2);
   free(tab1);
   free(tab2);
}
void drukuj(int n, int *tab)
{
   for(int i=0;i<n;i++)
       printf("%i: %i\n",i, tab[i]);
}
```

4 Programy wieloplikowe

Kod programu pisanego w języku C można podzielić na kilka osobnych. Zaleca się to realizować dla dłuższych kodów w celu poprawienia jego czytelności oraz kiedy chcemy odseparować pewne fragmenty kodu w celu ich późniejszego wykorzystania w innych projektach.

Podstawowy podział kodu w języku C jest na pliki źródłowe, z rozszerzeniem `*.c` i pliki nagłówkowe z rozszerzeniem `*.h`. W pliku nagłówkowym umieszcza się:

- podpięcie plików nagłówkowych bibliotek dyrektywą `#include<>`,
- dyrektyw preprocesora,
- deklaracji zmiennych i stałych globalnych,
- deklaracji funkcji.

W pliku z kodem źródłowym umieszcza się kod funkcji `main()` i pozostałych oraz na początku kodu umieszcza się jedną dyrektywę `#include "plik.h"` z wskazaniem pliku nagłówkowego. W momencie kompilacji zawartość wskazanego pliku zostanie w wstawiona w miejscu dyrektywy `include`. Korzystając z tej właściwości można na podobnych zasadach wydzielić do osobnego pliku wybrane funkcje, a następnie plik z nimi podpiąć do kodu programu dyrektywą `include`. Tak wydzielony zasób może łatwo zostać włączony do innego projektu.

Dzieląc kod na kilka plików istotne staje się zabezpieczenie programu przed wielokrotnym deklarowaniem zmiennych i funkcji o tych samych nazwach. Z tego powodu kod plików nagłówkowych zazwyczaj umieszcza się w układzie warunkowy pokazanym poniżej.

```
#ifndef PROGRAM_H
#define PROGRAM_H

kod nagłówkowy;

#endif
```

W pierwszej linii sprawdza się ze zdefiniowana została nazwa `"PROGRAM_H"`. Jeżeli nie to wykonany zostanie kod poniżej linii `#ifndef`, jeżeli zaś jest to kod nie zostanie wykonany. Druga linia definiuje nazwę `"PROGRAM_H"`. Dzięki temu kod jest zabezpieczony przed przypadkowym wielokrotnym wykonaniem kodu nagłówkowego.

program.c

```
---------------------------
#include "program.h"
int main()
{
    char c='A';
    printf("Program oblicza sume dwoch liczb.\n");
    int A = dane(c);
    int B = dane(c+1);
    printf("Suma liczb %i i %i wynosi %i\n",A,B,A+B);
}
---------------------------
```

```
program.h
--------------------------
#ifndef PROGRAM_H
#define PROGRAM_H
   #include <stdio.h>
   #include <stdlib.h>
   #include "funkcje.c"
   int dane(char);
   int wczytaj();
#endif
--------------------------
funkcje.c
--------------------------
int dane(char D)
{
    printf("Podaj liczbe %c=",D);
    return wczytaj();
}
int wczytaj()
{
    int x;
    scanf("%i",&x);
    return x;
}
--------------------------
```

Zadania

Na podstawie materiału omówionego w instrukcji napisać programy realizujące poniższe zadania.

- Program odnajdujący w przedziale od 0 do wartości podanej przez użytkownika wszystkie liczby pierwsze.
- Program sortujący w kolejności rosnącej lub malejącej (decyzja użytkownika) dowolnie długi ciąg liczb.

Programy napisać wykorzystując dynamiczną alokację pamięci, wydzielając do funkcji poszczególne zadania realizowane przez program. Kod programu podzielić na osobne pliki: plik nagłówkowy, plik główny z funkcją `main()` i plik z kodem pozostałych funkcji programu.