Podstawy Programowania
Zajęcia laboratoryjne 6
I rok Bioinformatyki Politechniki Poznańskiej

1 Wczytywanie z klawiatury, zabezpieczenia

Pisząc programy napotykamy często na błędy przy wczytywaniu znaków z klawiatury. Najczęstsze problemy to błąd konwersji (czytamy liczbę, a użytkownik podaje znak i program się zawiesza) lub program nie potrafi przeczytać ciągu znaków, jeśli ten zawiera spacje. Pojawiają się również problemy ze znakami nowej linii, które tylko czyhają w buforze i chcą zepsuć działanie naszego programu. Poniżej przykładowe rozwiązania tychże problemów.

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

int main()
{
    int i;
    char znak;
    printf("Podaj liczbe:");
    scanf("%d", &i);
    fflush(stdin); //czyszczenie bufora
    printf("Podaj znak:");
    scanf("%c", &znak);
    printf("Litera to: %c\n", znak);

    if (znak == 'C')
        printf("Wpisales C: %d ", i);
    else if (znak == 'F')
        printf("Wpisales F: %d ", i);
    else if (znak == 'K')
        printf("Wpisales K: %d ", i);

    return 0;
}
```

Powyższy kod (kod 1 dostępny w materiałach na stronie) pokazuje wykorzystanie funkcji fflush(stdin), która czyści bufor wejściowy po wczytaniu liczby do zmiennej i. W przypadku gdybyśmy pominęli wywołanie tej funkcji (wystarczy zakomentować linijkę) to znak enter ("\n") wczytywany jest w linii nr. 9. Przetestuj powyższy kod, zauważ co się stanie jeśli zakomentujesz funkcję fflush().

Poniższy kod (kod 2 dostępny w materiałach na stronie) przedstawia wykorzystanie funkcji getchar() oraz putchar() do wczytywania i wypisywania tekstu znak po znaku. Pętla while jest tak skonstruowana, że czyta wszystkie znaki, również spację i entery, jako kody ASCII (stąd zmienna jest typu int), a następnie wyświetla znak po znaku co zostało wpisane. Znaki są wypisywane z bufora dopiero po wpisaniu entera, tzn. enter kończy wykonanie funkcji getchar(). Natomiast pętla while zostaje zakończona dopiero po podaniu znacznika końca pliku EOF (end of file), co w poniższym przykładzie, jeśli nie czytamy z pliku, jest kombinacją klawiszy CTRL+Z (opcja ta nie działa w GDB Online).

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

```

```c
int main()
{
    int c;
    while((c = getchar()) != EOF)
    {
        printf(" Znak: ");
        putchar(c);
    }
    return 0;
}
```

Zmień powyższy kod tak aby pętla while zakończyła swoje działanie po podaniu znacznika końca pliku EOF lub znaku nowej linii.

Poniższy kod (kod 3 dostępny w materiałach na stronie) wykorzystuje wyrażenia regularne do zdefiniowania znaków które chcemy czytać. W poniższym przykładzie przeczytanych zostanie maksymalnie 280 znaków, które mogą się składać z liczb, spacji lub dużych i małych liter. Podanie innego znaku (np. ':') kończy wczytywanie tekstu z bufora.

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

int main()
{
    char bufor[280];
    printf("Napisz tweeta ponizej 280 znakow (bo inaczej sie zawiesze): ");
    scanf("%280[0-9a-zA-Z ]", bufor); //uwaga! miedzy 'Z' a ']' jest spacja!
    printf("Twoj tekst: %s", bufor);

    return 0;
}
```

2 Debugowanie (ang. debugging)

- Debugowanie jest procesem wyszukiwania oraz usuwania błędów znajdujących się w pisanym przez nas kodzie programu. Program wykorzystywany do tego celu nazywany jest debuggerem (ang. debugger – w wolnym tłumaczeniu odpluskiwacz). Istnieje szereg debuggerów w postaci komercyjnych rozwiązań zintegrowanych z pakietami Visual Studio, Visual C++ itd. Nie jesteśmy oczywiście zmuszeni do korzystania z powyższych środowisk. Uwagę swoją można bowiem skierować na narzędzia oparte na licencji publicznej GNU. Przykładem debuggera GNU jest program gdb. Zwykle nie musimy korzystać bezpośrednio z gdb gdyż narzędzia te są częścią zintegrowanego środowiska programistyczne (IDE), jak CodeBlocks czy Visual Studio.

- Aby skorzystać z przedstawionych metod wyszukiwania błędów konieczne jest, aby każdy testowany przez nas program uruchamiać w trybie "Debug" (nie wybierać opcji "Release"). W środowisku CodeBlocks ustawienie tej opcji można kontrolować w polu "Build target" na pasku "Compiler". Podobna sytuacja ma miejsce w przypadku środowiska Visual Studio.

  Uwaga, żeby skorzystać z opcji debugowania trzeba utworzyć projekt w CodeBlocks: File → New → Project:

  - W dalszej kolejności wybrać typ tworzonego projektu: Console application.
  - W wyświetlonym oknie dialogowym wybrać język w jakim będziemy programowali: C/C++
  - Wpisać nazwę projektu oraz jego lokalizację.
  - Naciśnięcie przycisku "Finish" kończy procedurę tworzenia projektu. W okienku "Menadżera" w zakładce "Projects" można przejrzeć drzewo przedstawiające strukturę plików w naszym projekcie. W chwili obecnej wyświetli się jeden plik *.c/.cpp o nazwie main.c/.cpp zawierający wygenerowany przez środowisko prosty kod programu, należy go nadpisać własnym.

- **Zadanie:** Wykorzystaj poniższy kod (kod 4 zawarty w materiałach na stronie) i skompiluj go w trybie "Debug". Zobacz jaki jest zakres widoczności zmiennych lokalnych i globalnych. Znajdź odpowiedź na

pytania: Czy funkcja print_table() wyświetla to samo w dwóch różnych wywołaniach? Dlaczego? Czy zmiany zmiennych licznik i argument w funkcji są później widoczne w dalszej części programu?

```c
#include<stdio.h>

int tab[10];
int licznik_globalny = 0;

int print_table(int argument)
{
    printf("Poczatek*** licznik: %d, argument: %d\n",licznik_globalny, argument);
    printf("Tabela: ");
    for (int i = 0; i < licznik_globalny; i++)
        printf("%d ", tab[i]);
    printf("\n");

    licznik_globalny = 2;
    argument = 2;
    printf("Koniec***  licznik: %d, argument: %d\n\n",licznik_globalny, argument);
}

int main() {

    int zmienna_lokalna = 5;
    int n = 10;

    for(licznik_globalny = 0; licznik_globalny < n; licznik_globalny++)
    {
       tab[licznik_globalny] = licznik_globalny * 2;
    }

    print_table(zmienna_lokalna);
    printf("Licznik: %d, zmienna_lokalna: %d\n\n", licznik_globalny, zmienna_lokalna);
    print_table(zmienna_lokalna);

    return 0;
}
```

- Kiedy dysponujemy już gotowym i działającym kodem rozwiązującym postawione zadanie, możemy przejść do ustawienia pierwszych punktów wstrzymania. Punktem wstrzymania (ang. breakpoint) jest pewne miejsce w programie w którym debugger automatycznie zatrzyma jego działanie. Nie jesteśmy oczywiście ograniczeni do pojedynczego punktu wstrzymania a ich położenie zależy wyłącznie od nas samych i wiedzy na temat możliwych błędów jakie mogą się pojawić w danym miejscu kodu źródłowego.
  - Ustaw kursor w linijce, w której ma znaleźć się punkt wstrzymania
  - Z menu wybierz: Debug → Toggle breakpoint (F5)
  - Uruchom program: Debug → Start (F8)
- Debugowanie:
  - Ustaw kilka punktów wstrzymania w różnych linijkach kodu.
  - Sprawdź działanie poleceń dostępnych w menu Debug:
  - Next Line (F7), przejście do następnej linii kodu źródłowego.
  - Next instruction (ALT+F7), przejście do następnej linii kodu maszynowego.
  - Step into (SHIFT+F7), wykonanie następnej instrukcji z wejściem do funkcji.
  - Step out (SHIFT+CTRL+F7), wyjście z funkcji.
  - Run to cursor (F4), przejście do pozycji wskazywanej przez kursor.
- Opcje debuggowania można znaleźć na pasku „Debugger” pod ikonką „Debugging windows”. Najbardziej nas będą interesowały następujące okna:

– Okno CPU Registers: okno to przedstawia aktualną zawartość rejestrów procesora. Możliwa jest bieżąca obserwacja zmian wartości zapisanych w rejestrach. Bezpośrednia manipulacja wartościami rejestrów procesora możliwa jest z poziomu języka maszynowego tj. assemblera.
– Okno Watches (zdecydowanie nas interesuje): okno to umożliwia podejrzenie bieżącej wartości zmiennych jak również śledzenie zmian ich wartości.

3 Zadania

Zad 1. Wykorzystaj kod 3 (kod dostępny w materiałach na stronie), aby zapoznać się z funkcjami atoi() oraz atof(), które służą do konwersji ciągu znaków do liczby całkowitej lub rzeczywistej. Zmodyfikuj kod 3 tak, aby była możliwość wczytywania tylko cyfr (zmodyfikuj wyrażenie regularne "[]"), a następnie przekonwertuj bufor do liczby całkowitej i liczby rzeczywistej oraz wyświetl wyniki.

Zad 2. Przetestuj zasięg zmiennych globalnych i lokalnych, w tym celu napisz program w którym: zadeklarujesz globalną zmienną a = 20 i globalną zmienną b = 10, napiszesz funkcję suma() zwracającą sumę a i b, a następnie w głównej części programu (main()) wywołasz funkcję suma() dla zmiennych globalnych, zadeklarujesz lokalna zmienną a = 1 i lokalną zmienną = 2 i wywołasz funkcję suma() dla zmiennych lokalnych. Zastanów się jak zasięg zmiennych wpływa na otrzymane wyniki. Jeśli masz wątpliwość to wykonaj debugowanie kodu.

Zad 3. Wykorzystaj kod 5 z protokołu z poprzednich zajęć dotyczący tablicy dwuwymiarowej i wykonaj jego debugowanie. Zwróć uwagę jak zmieniają się wartości iteratora i oraz j przy wyświetlaniu wartości macierzy. Czy rozumiesz konieczność użycia pętli w pętli?

Zad 4. Napisz program w którym zadeklarujesz globalną tablice o rozmiarze 10 (int tab[10]) i globalną zmienną rozmiar (int rozmiar = 10). W głównej funkcji programu wypełnij tę tablicę wartościami całkowitymi wprowadzonymi z klawiatury, następnie napisz własną funkcję która usuwa wartość z tablicy, oznacza to, że funkcja usun() prosi użytkownika o podanie pozycji [0,9] z której ma usunąć wartość. Poprzez usunięcie rozumiemy odpowiednie przesunięcie pozostałych wartości w tablicy i zmniejszenie globalnej zmiennej rozmiar o jeden. Wyświetl elementy tablicy po usunięciu.

Zad 5. Napisz program w którym zadeklarujesz 2 globalne tablice (float TEMP1[10], float TEMP2[10]) i globalny indeks (int INDEKS = 0) wspólny dla tych tablic, a następnie napisz 3 funkcje: przelicz(), zapisz(), wyswietl(), które wywołasz w głównej części kodu. Indeks globalny ma przechowywać numer ostatnio zapisanej pary temperatura_podana/temperatura_przeliczona. Funkcja przelicz() przyjmuje argument typu float (temperaturę) i zwraca przekonwertowaną temperaturę (ze skali C na skalę K, zgodnie ze wzorem: podana_temp + 273.15). Funkcja zapisz() wpisuje odpowiednie dane do tablic: temperatura podana przez użytkownika trafia do tablicy TEMP1, temperatura przeliczona trafia do tablicy TEMP2. Funkcja ta powinna przyjmować 2 argumenty wartość temperatury podanej przez użytkownika i wartość wyliczonej, ponadto funkcja zapisz() powinna sprawdzić czy jest miejsce w tablicy (jeśli jest to zapisać temperatury), a jeśli wielkość tablicy zostanie przekroczona to wyświetli się komunikat "Koniec miejsca w tablicy, przeliczenie nie zostanie zapisane". Funkcja wyswietl() wyświetla tylko tyle elementów ile zostało zapisanych w tablicach, zgodnie z wartością indeksu globalnego. Wyświetlenie elementów tablicy powinno być zgodne z formatem: nr. indeksu: temp_podana -> temp_przeliczona. Przetestuj napisane w funkcje w głównej funkcji main().

4 Zadanie domowe

Poniższe zadania dotyczą rozszerzenia do programu przeliczania temperatur.

Zad 1. Należy zadeklarować 4 tablice o wielkości 20 elementów, w których będą zapamiętywane wprowadzone przez użytkownika wartości temperatur, informacja o wykonanej konwersji (np. C -> K) oraz wynik obliczeń. Do tablicy dane będą wprowadzane na bieżąco za każdym razem gdy użytkownik wybierze z menu jakąkolwiek z pierwszych 6 opcji oznaczających przeliczanie temperatury. **Dane są wprowadzane do tablicy w funkcji** *Dodaj*(). **W głównej części programu (main) tablice nie mają być używane.**

- Będą potrzebne 4 tablice globalne: 2 na elementy typu float (o nazwach $t1$ i $t2$) oraz 2 na elementy typu char (o nazwach $z1$ i $z2$).
- Przykładowo, jeśli użytkownik wybierze przeliczanie temperatury w Kelwinach na Celsjusze i poda wartość temperatury 20 to w tablicy $t1$ zapamiętamy wartość 20, w tablicy $z1$ wartość 'K'. Natomiast w tablicy $t2$ zapamiętamy wartość wyniku w Celsjuszach czyli -253.15, a w tablicy $z2$ zapamiętamy literkę 'C'.
- Wszystkie tablice mają wspólny indeks - będzie to zmienna globalna *indeks* i będzie przechowywała numer ostatnio dodanej pary temperatura/przeliczenie jako aktualną wartość indeksu.
- Należy zaimplementować funkcję *Dodaj*, która będzie wpisywała dane do tablicy (4-argumentowa: wartość temperatury podanej przez użytkownika, wartość wyliczonej temperatury w zadanej skali, literki konwersji).
- Należy sprawdzić czy liczba elementów w tablicy nie przekracza jej wielkości (20 elementów). Jeśli tak to należy powiadomić użytkownika, że tablica jest pełna i żadne z jego kolejnych przeliczeń już się do niej nie zapiszą (program działa dalej, ale nie zapisuje już do tablicy).

Zad 2. Zamień opcje '7' w menu na opcję '0' - zakończenie działania programu.

Zad 3. Należy dodać opcję '7' do menu pozwalającą na wyświetlanie zawartości tablicy w trzech kolumnach, gdzie pojedyncza kolumna to np. #nr_indeksu 0C −> 32F. Wyświetlanie zawartości tablicy odbywa się we funkcji *Pokaz*(). Wyświetlane są tylko te elementy tablicy, które mają zapisane wartości (w tym celu wykorzystywana jest zmienna *indeks*).

Zad 4. Należy dodać opcję do menu '8' pozwalającą na usuwanie wiersza z tablicy po jego numerze. W ramach tej opcji użytkownik powinien zostać poproszony o numer wiersza do usunięcia, a następnie wywołać funkcję $Usun()$ podając jako argument numer wiersza do usunięcia. W funkcji $Usun()$ należy usunąć odpowiednie wartości ze wszystkich czterech tablic pod tym numerem indeksu.

- Wszystkie elementy w tablicy o numerach większych niż zadany należy przesunąć/przepisać o 1 w dół (jeśli usuwany element ma indeks 15 i jest więcej elementów w tablicy to 16-sty przepisujemy w miejsce 15-stego, 17-sty w miejsce 16-stego itd.), następnie zmniejszamy *indeks* o 1, pokazujący następne wolne miejsce w tablicy, w które można coś wpisać.
- Po usunięciu wiersza, program wyświetla tablicę (pomniejszoną o dany wiersz).
- Dodaj do opcji usuwania zabezpieczenie przed usunięciem wiersza z pustej tablicy, oraz przed usuwaniem wiersza większego niż *indeks*.

Zad 5. Należy dodać do menu opcje '9' pozwalającą na modyfikację wiersza:

- Modyfikacja inteligentna, program pozwala użytkownikowi na wprowadzenie modyfikacji, a następnie odpowiednio przelicza temperaturę. Przykładowo, jeśli w wierszu tablicy mamy: 0C -> 273K, to program pyta o podanie nowej temperatury i nowej skali temperatur, po wpisaniu np. 10F, program ma wpisać 10F zamiast 0C, a następnie zamiast 273K poprawnie przeliczyć temperaturę i wpisać ją w to miejsce.

- Modyfikacja wiersza dokonuje się w funkcji $Modyfikuj()$, po jej wywołaniu użytkownik jest poproszony o podanie pozycji do modyfikacji oraz nowej temperatury i jej skali. Pozostałe wartości w wierszu muszą zostać dostosowane nawet jeśli prowadzi to do sytuacji w której pojawi się przeliczenie: 0K -> 0K.

Zad 6. Należy dodać do menu opcję '10' pozwalającą na wypełnienie tablic losowymi wartościami (wartości muszą być sensowne).

- Utwórz funkcję $Losuj()$, w której użytkownik wybiera ile wierszy ma zostać wylosowanych. Jeśli liczba wierszy n zmieści się w tablicy ($n <=$ max wierszy $-$ aktualny indeks) to można losować tyle wierszy ile wybrał użytkownik. W przeciwnym wypadku należy ograniczyć liczbę losowanych wierszy n w taki sposób, aby zmieściły się w tablicy - należy poinformować użytkownika ile zmieści się nowych wierszy.
- W funkcji $Losuj()$ tablica powinna być wypełniana sensownymi wartościami: losowana jest pierwsza temperatura (zakres losowania określa użytkownik), losowana jest skala pierwszej temperatury C/K/F oraz na jaką skalę przelicza wartości C/K/F. Następnie wywołuje odpowiednią funkcję aby uzupełnić drugą temperaturę.

Zad 7. Należy dodać zabezpieczenia w programie, tak aby wprowadzenie innej liczby/znaku nie kończyło działania programu. Program wykonuje się w pętli, dopóki użytkownik nie wybierze opcji zakończenia programu. Menu powinno pojawiać się po każdej wykonanej operacji. Ponadto należy zapewnić przejrzystość kodu i przyjazność dla użytkownika.