Zajęcia B-1: “Własna arytmetyka”

Cel zajęć i efekty uczenia

Główne cele zajęć / materiał do opanowania:

- Zrozumienie i implementacja algorytmów pisemnego dodawania, [opcjonalnie] odejmowania i mnożenia
- Posługiwanie się typem *vector* jako alternatywą dla tablicy

Dodatkowe cele:

- Opanowanie typu *char*, świadomość różnicy między znakiem a jego kodem ASCII
- Konieczność przemyślenia algorytmu przed jego napisaniem, świadomość wartości dobrze zaprojektowanego algorytmu

Zadania do rozwiązania na sprawdzarce

Dodawanie
Dane są dwie duże (~500 000 cyfr) liczby w systemie dziesiętnym, obliczyć i wypisać ich sumę.

Odejmowanie *[opcjonalnie]*
Dane są dwie duże (~500 000 cyfr) liczby w systemie dziesiętnym, obliczyć i wypisać ich różnicę (odejmując mniejsza od większej).

[z reguły zadanie “Dodawanie” przeznaczone jest dla uczniów, którzy dopiero zaczynają naukę programowania, “Odejmowanie” dla uczniów z wcześniejszym doświadczeniem]

Mnożenie
Dane są dwie duże (~5000 cyfr) liczby w systemie dziesiętnym, obliczyć i wypisać ich sumę.

Plan zajęć

Szacunkowy czas trwania: 4 godziny lekcyjne.

1. Jak wczytać dużą liczbę z wejścia i jak ją przechowywać w pamięci
    - *Przypominamy, że typ int (a także “long long”) jest ograniczony i nie może przechowywać tak dużych liczb.*
    - *Liczby powinny być czytane jako łańcuchy znaków - wygodniej byłoby jednak trzymać je od najmniej znaczących cyfr do najbardziej znaczących*
2. Typ vector jako “tablica o zmiennej długości”, operacje na nim
    - *Wprowadzamy te operacje, które są nam potrzebne w danym momencie - deklaracja wektora (z rozmiarem i bez), metody size(), resize(), push_back(), pop_back().*
    - *Nie wchodzimy bez potrzeby w szczegóły implementacji wektora - jeśli uczniowie chcą wiedzieć, można opowiedzieć o podwajaniu, sygnalizując jednak, że dopiero w przyszłości dowiedzą się, dlaczego jest to efektywne.*

3. Konwersja łańcucha znaków do tablicy (wektora) liczb - kody ASCII, arytmetyka na znakach w C++
4. Algorytm pisemnego dodawania
    - *To jest w istocie prosty algorytm, uczniowie z reguły są go w stanie wymyślić zupełnie sami, warto skierować ich uwagę na mniej typowe przypadki (liczby różnej długości, suma dłuższa niż argumenty, zera wiodące w niektórych implementacjach).*
5. Algorytm pisemnego odejmowania
    - *Konieczność algorytmu w dwóch fazach - pierwsza pętla porównuje liczby, druga wykonuje odejmowanie*
    - *Odejmowanie jest bardzo podobne do dodawania - warto zwrócić uwagę, że "pożyczanie" możemy wykonać zawsze, nawet jeśli chwilowo pojawiłaby się cyfra -1 na którejś pozycji. Należy zwrócić uwagę na usunięcie zer wiodących i zabezpieczenie się na wypadek różnicy wynoszącej 0.*
6. Pisemne mnożenie - wariant "pomnóż liczbę A przez każdą cyfrę liczby B, dodaj wszystkie wyniki"
    - *Pokrótce opowiadamy ten wariant, zaznaczamy jednak, że będzie bardzo niewygodny w implementacji (oczywiście, chętni odważni uczniowie mogą się z nim zmierzyć)*
7. Pisemne mnożenie - wariant "pomnóż każdą cyfrę przez każdą, na końcu usuń przeniesienia"
    - *Ten wariant jest trudniejszy do zrozumienia, należy koniecznie najpierw zademonstrować go na kilku małych przykładach.*
    - *Warto podkreślać, że mimo trudności koncepcyjnej, ten wariant prowadzi do krótkiego kodu, w którym będziemy się rzadziej mylić - z punktu widzenia programisty jest znacznie lepszy.*