Politechnika Śląska w Gliwicach
Wydział Automatyki, Elektroniki i Informatyki

Programowanie Komputerów 3

Konsolowy arkusz kalkulacyjny

autor	Marcin Nycz
prowadzący	dr inż. Piotr Pecka
rok akademicki	2018/2019
kierunek	Informatyka
rodzaj studiów	SSI
semestr	3
termin laboratorium / ćwiczeń	Czwartek nieparzysty, 8:30-10:00
grupa	1
data oddania sprawozdania	2019-02-17

1 Treść zadania

Napisać program symulujący arkusz kalkulacyjny w wierszu poleceń systemu Windows

2 Analiza zadania

Podstawowym zadaniem arkusza kalkulacyjnego, jest możliwość wprowadzania danych do pól zwanych komórkami, w celu ich przechowywania, edycji oraz wykonywania na nich operacji takich jak dodawanie lub wyszukiwanie wartości największej. Aby stworzyć symulację takiego arkusza w wierszu poleceń, należy zapoznać się ze specyfiką obsługi konsoli w systemie Windows.

2.1 Struktury danych

W programie wykorzystano tablicę wskaźników do obiektów typu Cell, która przechowuje dane dotyczące zawartości danego arkusza. Arkusz kalkulacyjny posiada również pole do wpisywania danych do poszczególnych komórek.

3 Specyfikacja zewnętrzna

Dane wejściowe:
Program nie przyjmuje żadnych parametrów zewnętrznych, ale wczytuje dane o arkuszu z pliku o nazwie *data.txt*. Plik powinien znajdować się w bieżącym katalogu. W trakcie działania programu, dane z pliku zostaną pobrane i po ewentualnej modyfikacji zapisane z powrotem do tego samego pliku. Jeżeli plik *data.txt* nie zostanie odnaleziony, utworzy się pusty arkusz, w przeciwnym przypadku nastąpi załadowanie danych do arkusza.

Tryb wyboru komórki:
Po uruchomieniu programu wyświetlany jest arkusz kalkulacyjny. Na górze konsoli znajduje się pole do edycji danych, a pod nim tabela z komórkami. Każda komórka ma swój adres wiersza i adres kolumny. Adresy kolumn to litery alfabetu: A, B, C, ..., Y, Z, AA, AB, AC, itd. Adresy wierszy to kolejne liczby naturalne: 1, 2, 3, ..., 8, 9, 10, 11, itd.

Początkowo, kursor za pomocą którego wybierane są poszczególne komórki znajduje się na pozycji A1. Komórka, na której aktualnie znajduje się kursor, jest podświetlona (czarny tekst na białym tle). Aby przesunąć kursor należy posłużyć się **klawiszami strzałek**. Jeżeli przesuniemy się kursorem poza widoczną tabelę, zostaje ona przesunięta tak, aby kursor był ciągle widoczny (scrolling). Aby wejść w tryb edycji komórki, na której znajduje się kursor, należy nacisnąć klawisz **ENTER**.

Tryb edycji:
Po wejściu w tryb edycji, w polu do edycji danych pojawi się kursor konsolowy, załadowana zostaje zawartość wybranej komórki i możliwa jest edycja danych. Dane wprowadzane są z klawiatury. Aby usunąć znak, należy nacisnąć klawisz **BACKSPACE**. Aby wyjść z trybu edycji i zapisać wprowadzone dane należy ponownie nacisnąć klawisz **ENTER**.

Zakończenie pracy programu:
Aby zakończyć pracę programu, należy w trybie wyboru komórki nacisnąć klawisz **ESCAPE**. Arkusz zostanie zapisany do pliku o nazwie *data.txt* i program zakończy swoje działanie.

Jakie dane można wprowadzić?
Domyślnie, każda komórka jest komórką numeryczną, tj. komórką, która przechowuje liczby. Możliwe jest wprowadzenie liczb ułamkowych. Jeżeli do komórki numerycznej wprowadzimy znaki niebędące cyframi, w zawartości komórki wyświetli się komunikat `#STX_ERR`.
Przykładowe dane wprowadzane do komórki i wyświetlany rezultat:

```
10              →    10
42.3            →    42.3
Ala ma kota     →    #STX_ERR
Ala 2           →    #STX_ERR
2 Ala           →    2          (Jeżeli początek zawartości można
21,77           →    21         zinterpretować jako liczbę, to jest ona
                                interpretowana)
```

Aby wprowadzić tekst do komórki, należy na początku wpisać znak apostrofu. Zawartość komórki tekstowej powinna wyglądać np. tak:

```
‘10             →    10
‘Ala ma kota    →    Ala ma kota
‘Ala 2          →    Ala 2
‘2 Ala          →    2 Ala
```

Aby wykonać operacje na komórkach, należy utworzyć komórkę funkcyjną, która wykona daną operację i wyświetli wynik. Aby stworzyć komórkę funkcyjną, należy na początku wpisać znak równości. W aktualnej wersji programu istnieją dwie funkcje i trzy sposoby wprowadzania danych do komórek funkcyjnych.

Funkcje:
1.SUM - Suma
2.MAX - Znajdowanie wartości maksymalnej

Sposoby wprowadzania:
1. (A1:B4)
2. (A1,A3,B5,C8)
3. A4+C3

Komórki funkcyjne wykonają daną operację na sprecyzowanych komórkach i wyświetlą wynik. Ignorowane są komórki tekstowe, ale możliwe jest podanie innej komórki funkcyjnej, aby np. zsumować wynik poprzednich sumowań.
Poprawny format funkcji powinien wyglądać następująco:
=NAZWA_FUNKCJI(ARGUMENTY_FUNCJI)

Przykładowy sposób zastosowania komórek funkcyjnych:

=SUM(A1:C3) – Suma zawartości komórek z zakresu od A1 do C3, tj. komórki: A1, A2, A3, B1, B2, B3, C1, C2, C3
=SUM(A2,D3,W2) – Suma zawartości komórek A2, D3 i W2
=A3+G7 – Suma zawartości komórek A3 i G7
=MAX(R4:E1) – Znalezienie wartości maksymalnej z zakresu komórek E1 do R4.
=MAX(A3,A4,B2) - Znalezienie wartości maksymalnej z komórek A3, A4 i B2

Przykładowe niepoprawne funkcje wprowadzane do komórki:

=ALA(A1:A2) – brak funkcji ALA
=SUM (A1:A3) – Spacja między nazwą funkcji a nawiasem
=SUM (23:D2) – Niepoprawna nazwa komórki
=MAX(A1,D4 – Brak nawiasu zamykającego

Jeżeli w komórce funkcyjnej znajdzie się funkcja o niepoprawnym formacie, wyświetlany jest komunikat #NAME?

4 Specyfikacja wewnętrzna

4.1 Typy zdefiniowane w programie

W programie zdefiniowano następujące typy:

```
class Cell
{
protected:
	std::string contents;
public:
	Cell();
	Cell(std::string _contents);
	virtual void print() = 0;
	virtual std::string get_contents();
	virtual void set_contents(std::string _contents);
};
```

Podstawowa klasa definiująca komórkę. Zawiera pole typu *string* z zawartością danej komórki. *Cell* posiada konstruktor domyślny, ustawiający pole *contents* na puste, lub konstruktor z parametrem typu *string* kopiujący zawartość argumentu do pola *contents*. Jest to klasa abstrakcyjna, służąca jako baza do innych typów komórek.

```
class NumericCell: public Cell
{

public:
	using Cell::Cell;
	void print();
	double value();
};
```

Klasa pochodna klasy *Cell*. Definiuje komórkę numeryczną w arkuszu kalkulacyjnym. Na tym typie komórki wykonywane są operacje matematyczne. Komórka numeryczna jest komórką domyślną.

```
class TextCell: public Cell
{

public:
	using Cell::Cell;
	void print();
};
```

Klasa pochodna klasy *Cell*. Definiuje komórkę tekstową w arkuszu kalkulacyjnym. Aby dana komórka była traktowana jako komórka tekstowa, należy jako pierwszy znak zawartości wpisać apostrof.

```
class FunctionCell : public Cell
{
	std::string result;
	Cell* (*table)[MAX_ROWS][MAX_COLUMNS];
	double sum(int fcolumn, int frow, int lcolumn, int lrow);
	double sum(std::string list);
	double max(int fcolumn, int frow, int lcolumn, int lrow);
	double max(std::string list);
	void print_result();
public:
	FunctionCell(Cell*(*_table)[MAX_ROWS][MAX_COLUMNS]);
	FunctionCell(std::string _contents, Cell*(*_table)[MAX_ROWS][MAX_COLUMNS]);
	void print();
};
```

Klasa pochodna klasy *Cell*. Definiuje komórkę funkcyjną w arkuszu kalkulacyjnym. Jest ona odpowiedzialna za wykonywanie operacji na innych komórkach. Zawiera wskaźnik do tablicy, do której jest przypisana.

```
class Table
{
	CONSOLE_SCREEN_BUFFER_INFO offset;
	HANDLE hstdOut;
	Cell* tab[MAX_ROWS][MAX_COLUMNS];
	int column, row;
	int s_column, s_row;
public:
	Table();
	void print();
	void set_cursor();
	void add_to_row(int r);
	void add_to_column(int c);
	std::string get_current_cell_contents();
	void set_current_cell_contents(std::string _contents);
	int change_current_cell_type(std::string type);
	int save(std::string filename);
	int load(std::string filename);
};

std::string convert(int index);
```

Klasa definiująca tablicę wskaźników do komórek. Jest odpowiedzialna za przechowywanie informacji o kursorze i sposobie prezentacji tablicy użytkownikowi. Zmienne *column* i *row* odpowiadają współrzędnym aktualnie zaznaczonej komórki. Zmienne *s_colum* i *s_row* odpowiadają współrzędnej komórki, która znajduje się w lewym górnym rogu prezentowanej tablicy. Zmienne *offset* i *hstdOut* odpowiadają za zapamiętanie położenia tablicy w konsoli.

```
class TextField
{
	COORD position;
	std::string contents;
	void clear();
public:
	TextField();
	void enter(std::string _contents);
	void leave();
	void edit();
	std::string get_contents();
};

void set_cursor_visibility(int visible);
```

Klasa definiująca pole do wprowadzania danych przez użytkownika. Zawiera pole typu *string* z zawartością oraz pole typu *COORD* ze współrzędnymi określającymi położenie pola tekstowego na ekranie. Definiowana jest maksymalna długość tekstu wprowadzanego do pojedynczej komórki i wynosi ona `TEXT_FIELD_SIZE = 64`.

4.2 Struktura programu

W funkcji głównej najpierw tworzona jest tablica z komórkami i pole tekstowe. Wywoływana jest funkcja:

```
int Table::load(std::string filename);
```

która wczytuje dane do tablicy zapisane w pliku o danej nazwie.

Następnie tablica wyświetlana jest na ekranie, kursor ustawiany jest na komórkę A1 i rozpoczyna się główna pętla programu. Program za pomocą funkcji _kbhit() sprawdza, czy nie został przyciśnięty przycisk. Jeśli tak, to sprawdza się, jaki to był przycisk. W trybie wyboru komórki, wykrywane są jedynie przyciski strzałek, ENTER i ESCAPE. Za pomocą strzałek przesuwany jest kursor o jedną kolumnę lub wiersz za pomocą funkcji:

```
void Table::add_to_row(int r);
void Table::add_to_column(int c);
```

Następnie wyświetlana jest nowa pozycja kursora.
Jeśli przyciśnięty został klawisz ENTER, przerywana jest aktualna pętla programu. Wywoływane są funkcje pola tekstowego:

```
void TextField::enter(std::string _contents);
void TextField::edit();
```

ustawiające kursor w miejscu pola tekstowego i pozwalające na edycję danych. Po zakończeniu edycji sprawdzana jest zawartość pola tekstowego za pomocą funkcji:

```
std::string TextField::get_contents();
```

na podstawie której zmieniany jest typ danej komórki. Jeżeli na początku są znaki apostrofu lub równości, to komórka zmienia się na tekstową lub funkcyjną. W przeciwnym przypadku zostaje numeryczna. Dzieje się to za pomocą funkcji:

```
int Table::change_current_cell_type(std::string type);
```

Zawartość komórki jest ustawiana na tą, jaka znajdowała się w polu tekstowym, ponownie wyświetlana jest tablica oraz kursor i następuje powrót do początku pętli.

W przypadku naciśnięcia ESCAPE pętla programu jest przerywana, następuje zapisanie danych do pliku za pomocą funkcji

```
int Table::save(std::string filename);
```

i program kończy swoje działanie.

4.3 Szczegółowy opis implementacji funkcji

class Cell

```
virtual void print() = 0;
```
Funkcja abstrakcyjna, wirtualna. Istnieje po to, aby każda klasa pochodna, posiadała metodę służącą do wyświetlenia danej komórki na ekranie.

```
virtual std::string get_contents();
```
Funkcja zwraca zawartość komórki.

```
virtual void set_contents(std::string _contents);
```
Funkcja ustawia zawartość komórki na tą, która znajduje się w argumencie.

class NumericCell

```
void print()
```
Funkcja wyświetlająca zawartość komórki na ekranie. W przypadku gdy zawartość jest nieprawidłowa, np. jest to tekst, to wyświetlany jest komunikat `#STX_ERR`. Jeżeli liczba nie mieści się na ekranie, wyświetlane są znaki #.

```
double value();
```
Funkcja zwraca zawartość komórki w postaci zmiennej typu `double`.

class TextCell

```
void print();
```
Funkcja wyświetlająca zawartość komórki na ekranie. W przypadku gdy zawartość nie mieści się na ekranie, wyświetlana jest tylko jej część, wraz ze znakiem » .

class FunctionCell

```
double sum(int fcolumn, int frow, int lcolumn, int lrow);
double sum(std::string list);
double max(int fcolumn, int frow, int lcolumn, int lrow);
double max(std::string list);
```
Funkcje odpowiedzialne za wykonywanie operacji na komórkach. W wyniku zwracają rezultat operacji, a jako argumenty przyjmują początkową kolumnę i wiersz oraz końcową kolumnę i wiersz (zakres komórek, na których operacja ma być wykonana) lub też ciąg znaków wyszczególniający jakie komórki mają zostać poddane tej operacji.

```
void print_result();
```
Funkcja wypisująca wynik na ekranie.

```
void print();
```
Funkcja odpowiedzialna za sprawdzenie, czy format zawartości jest poprawny, jeżeli tak, to powoduje wywołanie odpowiedniej funkcji obliczającej wynik, a następnie wyświetlenie rezultatu operacji na ekranie.

class Table

```
int load(std::string filename);
int save(std::string filename);
```
Funkcje te odczytują i zapisują dane znajdujące się w tablicy. Dane zapisywane są w następujący sposób: Numer_wiersza Numer_kolumny Dane

```
void print();
```
Funkcja wyświetlająca tablicę na ekranie w miejscu przechowywanym w zmiennej `CONSOLE_SCREEN_BUFFER_INFO offset;`

```
void set_cursor();
```
Funkcja podświetlająca komórkę, na której aktualnie znajduje się kursor, jak i komórkę z nazwą kolumny i wiersza odpowiadającemu danej komórce.

```
void add_to_row(int r);
void add_to_column(int c);
```
Funkcje odpowiedzialne za inkrementację lub dekrementację bieżącej kolumny lub wiersza. W przypadku gdy bieżąca komórka nie mogłaby być zaprezentowana na ekranie, ponieważ wykroczyła poza obszar prezentowany, zmieniane są również zmienne *s_column* i *s_row*, aby podświetlona komórka była widoczna na ekranie (scrolling).

```
std::string get_current_cell_contents();
```
Funkcja zwraca zawartość bieżącej komórki.

```
void set_current_cell_contents(std::string _contents);
```
Funkcja ustawiająca zawartość bieżącej komórki, na tą podaną w argumencie.

```
int change_current_cell_type(std::string type);
```
Funkcja zmieniająca typ bieżącej komórki na odpowiedni, zależny od podanego parametru. Jeżeli na początku `type` są znaki apostrofu lub równości, to komórka zmienia się na tekstową lub funkcyjną. W przeciwnym przypadku staje się komórką numeryczną.

`class` TextField

```
void enter(std::string _contents);
```
Funkcja przywraca widoczność domyślnego kursora konsoli i ustawia zawartość pola tekstowego na tą znajdującą się w argumencie.

```
void leave();
```
Funkcja powoduje zniknięcie domyślnego kursora konsolowego i wyczyszczenie pola tekstowego.

```
void clear();
```
Funkcja powoduje wyczyszczenie pola tekstowego.

```
void edit();
```
Funkcja służąca obsłudze edycji zawartości pola tekstowego. Podobnie jak w funkcji *main* sprawdzane jest, czy nie został przyciśnięty klawisz. Jeżeli tak, to ręcznie dodawany, lub usuwany jest znak z ciągu znaków zawartości. W momencie przyciśnięcia ENTER, funkcja kończy swoje działanie.

```
std::string get_contents();
```
Funkcja zwraca zawartość pola tekstowego.

Pozostałe funkcje:

```
void set_cursor_visibility(int visible);
```
Funkcja ustawiająca widoczność domyślnego kursora konsolowego. Jeżeli argumentem jest 0, kursor staje się niewidoczny. W przeciwnym wypadku staje się widoczny.

```
std::string convert(int index);
```
Funkcja konwertująca indeks kolumny na odpowiedni ciąg znaków, np. indeks 1 na literę „A”, lub indeks 27 na ciąg „AB”.

```
std::string validate_cell(std::string& text);
```
Funkcja sprawdzająca, czy w podanym ciągu znaków na początku znajduje się poprawny adres komórki. Jeśli tak, to „ucina” go z `text` i zwraca go.
Np.
`text - A1,C3,Gsd123AlaMaKota`
Po przejściu przez funkcję zwrócona wartość to:
`A1`
a
`text - ,C3,Gsd123AlaMaKota`

```
int convert_column(std::string& name);
int convert_row(std::string& name);
```
Funkcje odpowiedzialne za konwersje adresu komórki na indeksy kolumn i wierszy, np. B3 na indeksy 2 i 3.

5 Testowanie

Program został przetestowany przez kilka osób, które dokonały dogłębnej analizy. Przetestowano zachowanie programu w przypadku próby naciśnięcia dowolnego przycisku w trybie wyboru komórki, jak i w trybie edycji. Zabezpieczono również komórki numeryczne w przypadku gdy liczba jest zbyt duża, aby mogła być zaprezentowana na ekranie lub gdy wynik operacji jest liczbą o dużej liczbie miejsc po przecinku.

6 Wnioski

Program symulujący arkusz kalkulacyjny w konsoli Windows okazał się być ciekawym wyzwaniem. Fakt, iż wcześniej miałem okazję pisać programy, w których należało wykorzystać uchwyty do konsoli, zdecydowanie ułatwił mi pracę nad projektem.

Problemem, który okazał się być jednym z najtrudniejszych do rozwiązania, było poprawne rozpoznawanie funkcji, oraz sprawienie, aby wyświetlany wynik był w odpowiednim formacie. Problem dotyczył również wyświetlania komórek numerycznych.

Innym problemem, który nie wydawał się zbyt trudny do rozwiązania, było poprawne wyświetlanie nazw kolumn, tj. sprawienie, aby nazwy kolumn były wyświetlane w sposób:

A, B, C, D, ... , Y, Z, AA, AB, AC, ..., AY, AZ, BA, BB, BC, itd.

Po długich zmaganiach z indeksami udało się pokonać problem, ale rozwiązanie zajęło zdecydowanie więcej czasu niż oczekiwałem.

W trakcie tworzenia funkcji obliczających sumę komórek należało pominąć komórki tekstowe. Ciekawym rozwiązaniem zastosowanym przeze mnie, było zastosowanie funkcji `typeid()`, która zwraca typ obiektu, na który pokazuje pewien wskaźnik. Pozwoliło mi to sprawdzenie, czy daną komórkę należy zsumować oraz na zrzutowanie typu Cell* na typ np. NumericCell* i zastosowanie funkcji należących jedynie do komórki numerycznej, np. `value()`.