Podstawy Programowania 1

Pliki

Arkadiusz Chrobot

Katedra Systemów Informatycznych

12 stycznia 2020

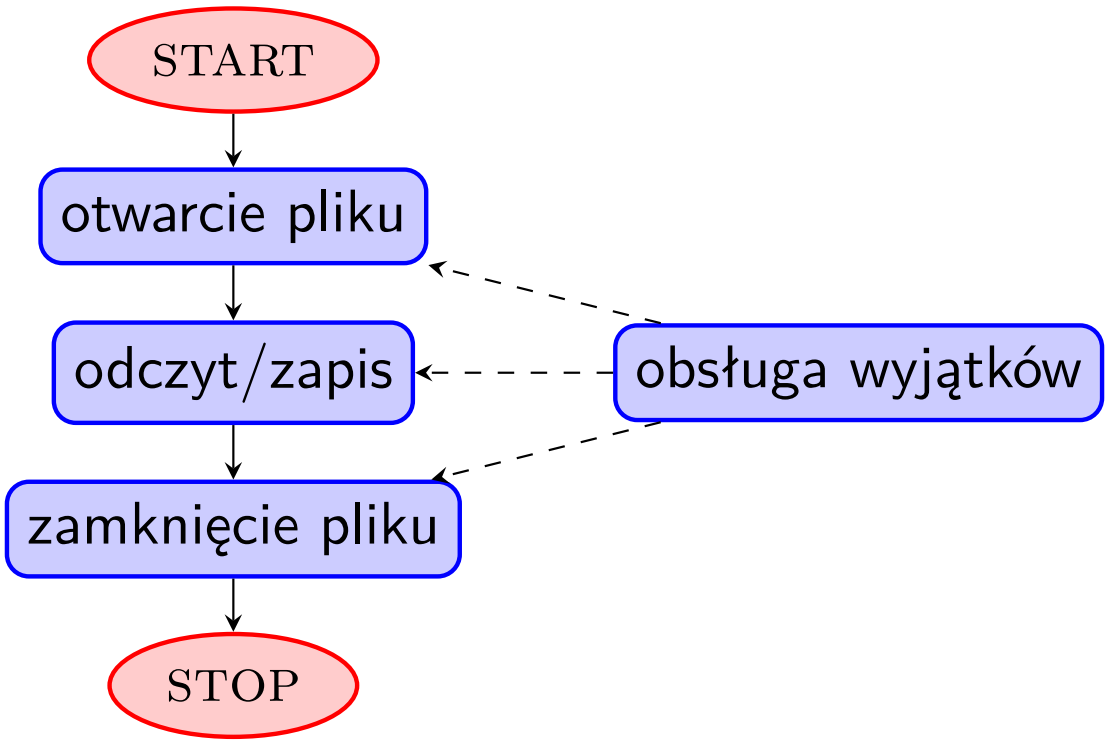
Plan

Wprowadzenie

Złożone programy komputerowe potrafią przetwarzać i generować ogromne ilości danych. Te informacje nie zawsze mieszczą się w pamięci operacyjnej komputera (RAM). Co więcej, po wyłączeniu zasilania zawartość tej pamięci zanika. Aby w sposób trwały przechowywać takie ilości informacji opracowano urządzenia pamięci masowej. Sposób przechowywania przez nie informacji zależy od ich typu. Aby ujednolicić sposób dostęp do danych gromadzonych w różnych rodzajach takiej pamięci stworzono specjalną strukturę danych, którą nazywa się *plikiem*.

Praca z plikiem

Pracę programu z plikiem można opisać następującym schematem blokowym:



Wyjątek jest każdą sytuacją, która wymaga odmiennego potraktowania przez program, bo może potencjalnie prowadzić do błędów przetwarzania. Przykładem wyjątku jest brak pliku, który program ma przeczytać.

Obsługa plików w języku C

Język C oferuje dla programistów dwie możliwości obsługi plików: niskopoziomową, związaną z tzw. deskryptorami plików i wysokopoziomową dostarczaną za pomocą tzw. strumieni. W ramach wykładu zostanie przedstawiona obsługa wysokopoziomowa, która gwarantuje między innymi automatyczne buforowanie informacji odczytywanych z i zapisywanych do pliku. Strumień jest zmienną typu `FILE *`[1]. Wszystkie funkcje związane z obsługą strumieni są zadeklarowane w pliku nagłówkowym `stdio.h`, który należy włączyć do programu.

[1] Dokładniej jest to wskaźnik na strukturę o nazwie `FILE`.

Standardowe strumienie

Zmienne typu `FILE *` służą w języku C nie tylko do obsługi plików. Mają one jeszcze kilka innych zastosowań. Po włączeniu do programu pliku nagłówkowego `stdio.h` stają się dostępne trzy zmienne tego typu o nazwach `stdin`, `stdout` i `stderr`. Pierwsza jest nazwana *standardowym strumieniem wejściowym* i jest domyślnie związana z klawiaturą, druga to *standardowy strumień wyjścia* i jest związana domyślnie z ekranem, trzecia to *standardowy strumień wyjścia diagnostycznego* i jest również domyślnie związana z ekranem. Do strumienia diagnostycznego programy powinny kierować komunikaty związane z pojawiającymi się wyjątkami lub błędami.

Otwieranie pliku

Otwieranie pliku jest czynnością rozpoczynającą pracę programu z plikiem. W języku C pozwala ona powiązać plik ze zmienną strumieniową i wykonywana jest poprzez wywołanie funkcji `fopen()`, która przyjmuje dwa argumenty. Pierwszym jest łańcuch znaków będący ścieżką do otwieranego pliku, a drugim ciąg, który może się składać z następujących znaków:

Tryb	**Opis**
`r`	Otwarcie tylko do odczytu. Wskaźnik pliku wskazuje na jego początek.
`r+`	Otwarcie do odczytu i zapisu. Wskaźnik pliku wskazuje na jego początek.
`w`	Utworzenie pliku i otwarcie tylko do zapisu. Jeśli plik o podanej nazwie istnieje, to jego zawartość jest kasowana. Wskaźnik pliku wskazuje na jego początek.
`w+`	Jak wyżej, ale z możliwością odczytu.
`a`	Otwarcie do dopisywania. Wskaźnik pliku wskazuje na jego koniec. Jeśli plik nie istnieje, to będzie utworzony.

Otwieranie pliku

Kontynuacja

Tryb	Opis
a+	Jak wcześniej, ale z możliwością odczytu.
b	Otwarcie będzie dotyczyło pliku binarnego. Obecnie większość systemów operacyjnych ignoruje ten znacznik.

Znaki z tabeli mogą być łączone ze sobą, o ile nie wykluczają się wzajemnie (np. otwarcie do zapisu i dopisywania nie mogą być użyte razem). Znacznik `b` informował funkcję, że otwierany plik będzie plikiem binarnym, a nie tekstowym. Obecnie tylko niektóre systemy operacyjne robią takie rozróżnienie. Wynik działania funkcji `fopen()` należy przypisać do zmiennej typu `FILE *`. Jeśli ta zmienna po wykonaniu czynności otwierania pliku będzie miała wartość `NULL`, to oznacza to, że wystąpił wyjątek i z jakiś przyczyn nie udało się otworzyć pliku.

Obsługa wyjątków

Jeśli funkcja `fopen()` zasygnalizowała wyjątek, to można poznać kod tego wyjątku za pomocą wywołania funkcji `ferror()`. Przyjmuje ona jako argument wywołania zmienną typu `FILE *`, a zwraca kod błędu lub zero, jeśli nie było żadnego wyjątku. Ten kod może zostać wyzerowany przy pomocy wywołania funkcji `clearerr()`, ale należy jej używać ostrożnie, bo zeruje ona również znacznik końca pliku. Podobnie jak `ferror()` przyjmuje ona jako argument wywołania strumień. Obie funkcje działają nie tylko dla wyjątków zgłaszanych przez `fopen()`, ale również przez inne funkcje operujące na strumieniach. Opis słowny (w języku zależnym od konfiguracji komputera) można otrzymać za pomocą funkcji `perror()`. Należy ją wywołać tuż po zakończeniu funkcji, jeśli zgłosiła ona wyjątek i jako argument wywołania przekazać jej nazwę tej funkcji w postaci łańcucha znaków.

Zapis do pliku

Zapis do pliku tekstowego

Zapis do pliku tekstowego[2] możemy zrealizować przy pomocy funkcji `fputc()`, która przyjmuje dwa argumenty wywołania - kod ASCII zapisywanego znaku (jako wartość typu `int`) oraz strumień związany z plikiem do którego nastąpi zapis. Zwraca ona kod zapisanego znaku lub stałą o nazwie `EOF` (ang. *End Of File*) w przypadku niepowodzenia. Cały łańcuch znaków, oprócz znaku `'\0'`, można zapisać do pliku przy pomocy funkcji `fputs()`, która przyjmuje dwa argumenty wywołania - tablicę znaków, z ciągiem do zapisania i strumień. Funkcja ta zwraca liczbę zapisanych do pliku znaków lub opisaną wyżej stałą w razie niepowodzenia.

[2] Plik tekstowy to taki plik, który zawiera wartości zapisane w postaci kodów ASCII, a więc można jego zwartość przeczytać i edytować dowolnym edytorem tekstowym.

Zapis do pliku

Zapis do pliku tekstowego - kontynuacja

Dane różnych typów można skonwertować do łańcucha znaków i zapisać je w pliku tekstowym za pomocą funkcji `fprintf()`, która wywoływana jest podobnie do funkcji `printf()`, ale jako pierwszy argument przyjmuje strumień związany z plikiem, do którego mają trafić dane. Funkcja `fprintf()` zwraca liczbę znaków, które udało się zapisać w pliku[3]. Jest ona także stosowana do zapisu komunikatów o wyjątkach do standardowego wyjścia diagnostycznego.

[3] Ciekawostką jest to, że funkcja `printf()` zwraca liczbę wypisanych na ekran znaków, ale zazwyczaj ta informacja jest ignorowana w programie.

Zapis do pliku

Zapis do pliku binarnego

Jeśli chcemy zapisać informację do pliku binarnego[4] to możemy w tym celu użyć funkcji `fwrite()`, która przyjmuje cztery argumenty wywołania - wskaźnik na zmienną, która zawiera informację do zapisania[5], rozmiar pojedynczej porcji danych zapisywanych z bufora, liczbę tych porcji i strumień. Ta funkcja zwraca liczbę faktycznie zapisanych porcji. Jeśli jest ona mniejsza od wartości przekazanej jako jej trzeci argument wywołania, to znaczy, że wystąpił wyjątek - błąd zapisu.

[4] W pliku binarnym wartości są zapisywane w takiej samej postaci, jak w pamięci operacyjnej, czyli jako ciąg bitów, które niekoniecznie są kodami ASCII.

[5] Taką zmienną nazywa się *buforem*.

Odczyt z pliku

Podstawowym problemem podczas odczytu z pliku jest określenie kiedy skończą się w nim dane. W języku C, w przypadku obsługi plików za pomocą strumieni, o osiągnięciu końca pliku informuje funkcja `feof()`. Przyjmuje ona jako argument wywołania strumień, a zwraca liczbę typu `int`. Jeśli ta liczba jest różna od zera, to znaczy, że w pliku nie ma więcej danych i należy przerwać jego odczytywanie. Wywołanie tej funkcji jest często używane w warunkach pętli, co zostanie zademonstrowane w przykładowych programach.

Odczyt z pliku

Odczyt z pliku tekstowego

Pojedyncze znaki z pliku tekstowego można odczytywać przy pomocy funkcji `fgetc()`, która przyjmuje jako argument wywołania strumień, a zwraca kod ASCII odczytanego znaku jako liczbę typu `int` lub stałą `EOF` w przypadku wystąpienia wyjątku lub w przypadku osiągnięcia końca pliku. Odczytu łańcucha znaków z pliku można dokonać z użyciem funkcji `fgets()`, która przyjmuje trzy argumenty - tablicę znaków, w której będzie umieszczony odczytany z pliku ciąg znaków, rozmiar tej tablicy (funkcja odczytuje o jeden znak mniej niż wynosi ten rozmiar) oraz strumień, z którego ma być dokonany odczyt. Jeśli dane różnych typów zostały zapisane z użyciem funkcji `fprintf()`, to można je odczytać używając funkcji `fscanf()`, która musi być wywołana z użyciem odpowiedniego ciągu formatującego. Jej pierwszym argumentem jest strumień, tak jak w przypadku `fprintf()`. Pozostałe argumenty, oprócz ciągu formatującego, są wskaźnikami na zmienne dla danych. Funkcja ta zwraca liczbę elementów pliku, które udało się dopasować do ciągu formatującego. W przypadku odczytu zakończonego niepowodzeniem zwraca ona stałą `EOF`.

Odczyt z pliku

Odczyt z pliku binarnego

Dane zapisane przy pomocy funkcji `fwrite()` można odczytać z pliku binarnego za pomocą funkcji `fread()`. Przyjmuje ona takie same argumenty, jak funkcja wspomniana wcześniej, z tym, że w jej przypadku dane są odczytywane ze strumienia i umieszczane w zmiennej będącej jej pierwszym argumentem wywołania. Zwraca ona liczbę porcji danych, które faktycznie udało się jej odczytać z pliku. Jeśli ta liczba będzie mniejsza od liczby przekazanej jej jako trzeci argument wywołania, to oznacza to, że w pliku nie ma już więcej danych do odczytu.

Zamykanie pliku

Funkcją służącą do zamykania pliku jest `fclose()`. Jako argument wywołania przyjmuje ona strumień związany z plikiem, który ma zostać zamknięty. Zwraca ona wartość różną od zera, jeśli jej działanie zakończyło się niepowodzeniem.

Inne funkcje do obsługi plików

Z każdym plikiem obsługiwanym za pomocą strumieni związany jest *wskaźnik pliku*, który pełni podobną rolę jak indeks w tablicy i o którym wspomniano w tabeli znaczników trybów dostępu dla funkcji `fopen()`. Funkcje zapisujące i odczytujące pliki niejawnie (automatycznie) zwiększają jego wartość. Taki rodzaj dostępu do danych, w który wartość wskaźnika jest wyłącznie zwiększana nazywamy *sekwencyjnym*. Sposób *swobodny* lub inaczej *bezpośredni* umożliwia zarówno zwiększanie, jak i zmniejszanie wskaźnika pliku, a więc posługiwanie się plikiem podobnie do tablicy. Modyfikacja wartości wskaźnika pliku wykonywana jest za pomocą funkcji `fseek()`, która przyjmuje trzy argumenty wywołania. Pierwszym jest strumień związany z plikiem, którego wskaźnik ma być przesunięty, drugim jest liczba bajtów (typu `long int`), o którą ma być przesunięty, a trzecim jedna z trzech stałych: `SEEK_SET` - przesunięcie będzie liczone względem początku pliku, `SEEK_CUR` - przesunięcie będzie liczone względem bieżącej pozycji wskaźnika, `SEEK_END` - przesunięcie będzie liczone względem końca pliku. Wyjątek jest sygnalizowany przez funkcję zwróceniem wartości `-1`.

Inne funkcje do obsługi plików

Funkcja `rewind()` przesuwa wskaźnik pliku na jego początek. Jako argument przyjmuje strumień i nie zwraca żadnej wartości. Funkcja `ftell()` również przyjmuje strumień jako argument wywołania i zwraca bieżącą wartość wskaźnika pliku (liczba typu `long int`). Do manipulowania wskaźnikiem pliku można użyć także funkcji `fgetpos()` i `fsetpos()`, które nie będą jednak tutaj dokładniej opisywane.

Inne funkcje do obsługi plików

Podczas zapisu dane nie zawsze trafiają bezpośrednio do pliku, ale mogą być umieszczone w pamięci operacyjnej w buforze, który program niejawnie przeznacza na nie. Dzieje się tak dlatego, że operacja zapisu danych do urządzenia pamięci masowej trwa stosunkowo długo, więc aby nie blokować swojego działania program odkłada ją na później. Aby opróżnić te bufory można użyć funkcji `fflush()`, która przyjmuje jako argument wywołania strumień, a zwraca zero w przypadku powodzenia wykonywanej przez nią czynności lub stałą `EOF` jeśli wystąpił wyjątek. Opróżnienie tych buforów nie gwarantuje jednak zapisu danych na nośnik, bo buforowanie może stosować także system operacyjny, który nadzoruje wykonanie programu. Bufory te są automatycznie opróżniane po wywołaniu `fclose()`. Standardowa biblioteka języka C dostarcza także innych funkcji do zarządzania tymi buforami, ale nie będą one na tym wykładzie omawiane.

Funkcje zarządzające plikami

W standardowej bibliotece języka C zostały zdefiniowane także funkcje, które służą do zarządzania plikami. Opisane zostaną tu dwie z nich. Funkcja `remove()` (w uproszczeniu) służy do usuwania plików lub katalogów. Jako argument wywołania przyjmuje ona łańcuch znaków będący nazwą usuwanego pliku (katalogu). Jeśli wystąpi wyjątek, to funkcja ta zwróci liczbę `-1`. Funkcja `rename()` zmienia nazwę pliku lub jego lokalizację na nośniku danych. Przyjmuje ona dwa argumenty. Pierwszym jest stara nazwa (lokalizacja) pliku, a drugim nowa nazwa (lokalizacja). Funkcja zwraca zero, jeśli czynność zmiany nazwy (lokalizacji) się powiodła lub `-1` w przeciwnym przypadku.

Przykład pierwszy

Zapis pojedynczych znaków do pliku tekstowego

Jako pierwszy przykład zostanie zaprezentowany program, który zapisuje 100 wylosowanych małych liter do pliku tekstowego, a następnie je z niego odczytuje i wypisuje na ekran. W programie zaprezentowano także jeden z możliwych sposobów wykrywania i informowania o wyjątkach. Ich obsługa nie jest jednak pełna, np. otwarty plik nie jest zamykany, jeśli nastąpił wyjątek jego zapisu.

Przykład pierwszy

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

#define LENGTH 100
```

Przykład pierwszy

Komentarz

Fragment kodu źródłowego programu zaprezentowany na poprzednim slajdzie zawiera instrukcje włączające pliki nagłówkowe. Oprócz pliku `stdio.h` włączane są także pliki `stdlib.h` oraz `time.h` ponieważ program będzie korzystał z generatora liczb pseudolosowych. Stała `LENGTH` określa liczbę elementów tablic znaków, w których będą zapisane komunikaty diagnostyczne.

Pierwszy przykład

```
void display_exception_message(int code)
{
    char exception_description [][LENGTH] = {
        "Błąd otwarcia pliku do zapisu.",
        "Błąd zapisu pliku.",
        "Błąd zamknięcia pliku.",
        "Błąd otwarcia pliku do odczytu."
    };
    fprintf(stderr,"%s\n",exception_description[-code-1]);
}
```

Przykład pierwszy

Komentarz

Zaprezentowana na poprzednim slajdzie funkcja będzie służyła do wypisywania na ekran komunikatów związanych z wyjątkami, które mogą pojawić się podczas działania innych funkcji zawartych w tym programie. Treści tych komunikatów zawarte są w tablicy `exception_description`. Do ich wypisania używana jest funkcja `fprintf()`, która jako pierwszy argument przyjmuje strumień diagnostyczny. Proszę zwrócić uwagę na wyliczanie indeksu dla tablicy komunikatów. Ponieważ wyjątki będą sygnalizowane ujemnymi liczbami, poczynając od `-1`, to aby otrzymać prawidłowy indeks zmieniany jest znak kodu błędu i odejmowana jest od niego liczba jeden. Tak więc kod `-1` będzie oznaczał wyjątek związany z otwarciem pliku, `-2` z zapisem do pliku itd.

Przykład pierwszy

```
int fill_file(char *file_name)
{
    FILE *file = NULL;
    srand(time(0));
    file = fopen(file_name,"w");
    if(file==NULL)
        return -1;
    int i;
    for(i=0; i<100; i++)
        if(fputc('a'+rand()%('z'-'a'+1),file)==EOF)
            return -2;
    if(fclose(file)!=0)
        return -3;
    return 0;
}
```

Przykład pierwszy

Komentarz

Funkcja `fill_file()` odpowiedzialna jest za zapis stu małych liter do pliku tekstowego. Posiada ona jeden parametr, przez który przekazywana jest nazwa pliku do zapisu, a jako jej wartość zwracany jest kod wyjątku. W przypadku, gdy będzie on równy zero będzie to oznaczało, że wszystkie czynności wykonywane przez funkcję zakończyły się prawidłowo. Jeśli będzie różny od zera, to znaczy, że wystąpił gdzieś wyjątek. Na początku funkcja inicjuje zmienną strumieniową, a następnie generator liczb pseudolosowych. Następnie otwiera ona plik do zapisu. Jeśli on nie istniał to jest tworzony, w przeciwnym przypadku kasowana jest jego dotychczasowa zawartość. Jeżeli nie powiodło się otwarcie pliku, to funkcja zwraca liczbę `-1` i kończy swoje działanie. Jeśli jednak ta operacja przebiegła prawidłowo, to funkcja w pętli losuje sto małych liter i zapisuje je do pliku, za każdym razem badając, czy nie wystąpił wyjątek zapisu. Jeśli by się tak stało, to funkcja przerwie swoje wykonanie i zwróci liczbę `-2`. Po zakończeniu pętli funkcja zamyka plik. Jeśli ta czynność się nie powiedzie, to jest to sygnalizowane przez nią zwróceniem wartości `-3`. Funkcja kończy się zwracając liczbę `0`.

Przykład pierwszy

```
int read_file(char *file_name)
{
    FILE *file = fopen(file_name,"r");
    if(file==NULL)
        return -4;
    while(!feof(file)) {
        char data_from_file = fgetc(file);
        if(data_from_file!=EOF)
            printf("%c ",data_from_file);
    }
    if(fclose(file)!=0)
        return -3;
    return 0;
}
```

Pierwszy przykład

Komentarz

Funkcja `read_file()` odczytuje wszystkie znaki z pliku, którego nazwa jest jej przekazywana przez parametr i wyświetla je na ekran. Najpierw otwiera ona plik do odczytu. Jeśli ta operacja się nie powiedzie, to funkcja zwraca wartość `-4` i kończy swoje działanie. W przeciwnym przypadku w pętli `while` odczytuje ona plik znak po znaku z użyciem funkcji `fgetc()` i wypisuje te znaki na ekranie. Zapis `!feof(file)` w warunku pętli jest skróconą wersją zapisu `feof(file)==0`. W pętli dodatkowo badane jest, czy `fgetc()` nie zwróciła wartości `EOF`. Ta wartość sygnalizuje, że przeczytany został znak końca pliku i nie trzeba nic już wypisywać na ekranie. Po zakończeniu pętli funkcja zamyka plik. Jeśli nie uda się wykonać prawidłowo tej czynności, to `read_file()` przerywa swoje działanie i zwraca ten sam kod wyjątku, co funkcja `fill_file()`. Jeśli wszystkie czynności wewnątrz funkcji przebiegły prawidłowo, to kończąc się zwraca ona liczbę zero.

Przykład pierwszy

```
int main(void)
{

    int result = fill_file("test.txt");
    if(result<0)
        display_exception_message(result);
    result = read_file("test.txt");
    if(result<0)
        display_exception_message(result);
    return 0;
}
```

Przykład pierwszy

Komentarz

W funkcji `main()` najpierw wywoływana jest funkcja `fill_file()`, a następnie `read_file()`. Wynik wykonania każdej z nich zapisywany jest w zmiennej `result`. Jeśli będzie on ujemny, to wówczas wywoływana jest funkcja `display_exception_message()`, do której przekazywana jest wartość kodu wyjątku.

Przykład drugi

Drugi program pozwala zapisać do pliku tekstowego łańcuchy znaków, które mogą zawierać spacje i inne znaki. Te ciągi znaków wprowadzane są przez użytkownika do momentu, aż nie poda on wyrazu „stop”. Podobnie jak w poprzednim programie obsługa wyjątków jest tylko częściowa.

Przykład drugi

```
#include<stdio.h>
#include<string.h>

#define LENGTH 100
#define SENTENCE_LENGTH 81
```

Przykład drugi

Komentarz

Oprócz pliku nagłówkowego `stdio.h` do programu włączany jest również plik `string.h`, ponieważ w programie używane są funkcje realizujące operacje na łańcuchach znaków. Znaczenie stałej `LENGTH` jest takie samo jak w poprzednim programie. Stała `SENTENCE_LENGTH` definiuje wielkość tablicy znaków, w jakiej będą zapamiętane ciągi znaków zapisywane w pliku. Osiemdziesiąt to najczęściej spotykana maksymalna liczba znaków w wierszu ekranu.

Przykład drugi

```
void display_exception_message(int code)
{
    char exception_description [][LENGTH] = {
        "Błąd otwarcia pliku do zapisu.",
        "Błąd zapisu pliku.",
        "Błąd zamknięcia pliku.",
        "Błąd otwarcia pliku do odczytu."
    };
    fprintf(stderr,"%s\n",exception_description[-code-1]);
}
```

Przykład drugi

Komentarz

Funkcja `exception_message()` jest zaimplementowana dokładnie tak samo jak w poprzednim programie.

Przykład drugi

```
int write_sentences_to_file(char *file_name)
{
    char sentence[SENTENCE_LENGTH] = "\0";
    FILE *file=fopen(file_name,"w");
    if(file==NULL)
        return -1;
    while(strncmp(sentence,"stop",SENTENCE_LENGTH-1)!=0)
    {
        scanf("%80[^\n]s",sentence);
        while(getchar()!='\n');
        if(fprintf(file,"%s\n",sentence)!=strlen(sentence)+1)
            return -2;
    }
    if(fclose(file)!=0)
        return -3;
    return 0;
}
```

Przykład drugi

Komentarz

Funkcja `write_sentences_to_file()` zapisuje pobrane od użytkownika ciągi znaków do pliku, którego nazwa jest przekazana jej przez parametr. W funkcji deklarowana jest zmienna lokalna o nazwie `sentence`, która jest tablicą znaków. Jest ona inicjowana pustym łańcuchem znaków. Funkcja najpierw otwiera plik do zapisu. Jeśli ta operacja zakończy się sukcesem to w pętli pobierany jest od użytkownika łańcuch znaków i zapisywany do pliku za pomocą funkcji `fprintf()`. Kontrola poprawności zapisu polega na sprawdzeniu liczby znaków, które ta funkcja zapisała z liczbą znaków zapisywanego ciągu, przy uwzględnieniu, że do pliku zapisywany jest dodatkowo znak nowego wiersza. Wewnętrzna pętla `while` służy do usunięcia znaku nowego wiersza (klawisza `Enter`) ze standardowego wejścia, aby funkcja `scanf()` mogła prawidłowo pobrać kolejny ciąg od użytkownika. Po wprowadzeniu przez niego wyrazu „stop” pętla jest kończona, a plik jest zamykany. Jeśli któraś z operacji na pliku zakończy się niepowodzeniem to funkcja przerywa swoje działanie i zwraca kod wyjątku. Jeśli wszystkie czynności zakończą się pomyślnie, to funkcja kończąc się zwróci liczbę zero.

Przykład drugi

```
int read_sentences_from_file(char *file_name)
{
    char sentence[SENTENCE_LENGTH] = "\0";
    FILE *file = fopen(file_name,"r");
    if(file==NULL)
        return -4;
    while(feof(file)==0) {
        fscanf(file,"%[^\n]s",sentence);
        if(feof(file)==0) {
            puts(sentence);
            while(fgetc(file)!='\n');
        }
    }
    if(fclose(file)!=0)
        return -3;
    return 0;
}
```

Przykład drugi

Komentarz

Funkcja `read_sentence_from_file()` odczytuje wiersz po wierszu zapisane w pliku tekstowym przez poprzednio opisywaną funkcję ciągi znaków i wypisuje je na ekranie. Przez parametr przekazywana jest jej nazwa pliku tekstowego. Podobnie jak w jej poprzedniczce jest w niej zadeklarowana lokalna tablica znaków. Funkcja najpierw otwiera plik do odczytu, a następnie w pętli odczytuje kolejne wiersze pliku i wypisuje je na ekran. Po każdym wykonaniu funkcji `fscanf()`, jeśli nie wystąpił jeszcze znacznik końca pliku, wywoływana jest w wewnętrznej pętli funkcja `fgetc()`, celem odczytania znaków końca wiersza i przesunięcia wskaźnika pliku na właściwe dane, które będą odczytywane przez `fscanf()` w kolejnej iteracji zewnętrznej pętli. Po zakończeniu odczytu funkcja zamyka plik. Podobnie jak w przypadku poprzednio opisywanej funkcji każdy wyjątek powoduje zakończenie przez funkcję działania i zwrócenie odpowiedniego kodu wyjątku. Pomyślne ukończenie działania funkcja sygnalizuje zwracając liczbę zero.

Przykład drugi

```
int main(void)
{
    int result = write_sentences_to_file("test.txt");
    if(result!=0)
        display_exception_message(result);
    result = read_sentences_from_file("test.txt");
    if(result!=0)
        display_exception_message(result);
    return 0;
}
```

Przykład drugi

Komentarz

Funkcja `main()` jest zaimplementowana podobnie, jak w poprzednio opisywanym programie.

Przykład trzeci

Trzeci program zapisuje do pliku binarnego struktury zawierające wylosowane współrzędne punktów w trójwymiarowym układzie kartezjańskim. Również i tym razem obsługa wyjątków jest niepełna, aby nie komplikować zbytnio zapisu programu. Dodatkowo w programie zostanie zaprezentowana operacja dopisywania danych na końcu istniejącego pliku.

Przykład trzeci

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

#define LENGTH 100
```

Przykład trzeci

Komentarz

Początek programu jest taki sam, jak w przypadku programu pierwszego.

Przykład trzeci

```
void display_exception_message(int code)
{
    char exception_description[][LENGTH] = {
        "Błąd otwarcia pliku do zapisu.",
        "Błąd zapisu pliku.",
        "Błąd zamknięcia pliku.",
        "Błąd otwarcia pliku do dopisywania.",
        "Błąd dopisywania do pliku",
        "Błąd otwarcia pliku do odczytu"
    };
    fprintf(stderr,"%s\n",exception_description[-code-1]);
}
```

Przykład trzeci

Komentarz

Tablica `exception_description` w funkcji `display_exception_message()` zawiera więcej elementów niż jej odpowiedniczki z poprzednich programów. Dodatkowe komunikaty związane są z wyjątkami, jakie mogą pojawić się podczas operacji dopisywania danych do istniejącego pliku.

Przykład trzeci

```
struct coordinates {
    double x,y,z;
};
```

Przykład trzeci

Komentarz

Zaprezentowany na poprzednim slajdzie typ strukturalny definiuje struktury, które będą zapisywane do pliku binarnego.

Przykład trzeci

```
int write_to_file(char *file_name)
{
    FILE *file = fopen(file_name,"w");
    if(file==NULL)
        return -1;
    int i;
    for(i=0; i<10; i++) {
        const int RANGE = 20;
        struct coordinates point;
        point.x = rand()%RANGE;
        point.y = rand()%RANGE;
        point.z = rand()%RANGE;
        if(fwrite(&point,sizeof(point),1,file)!=1)
            return -2;
    }
    if(fclose(file)!=0)
        return -3;
    return 0;
}
```

Przykład trzeci

Komentarz

Funkcja `write_to_file()` wypełnia pola struktury zadeklarowanej jako zmienna lokalna liczbami wylosowanymi z zakresu od 0 do 19, a następnie zapisuje tę strukturę do pliku wywołując w tym celu funkcję `fwrite()`. Czynność tę powtarza 10 razy. Poprawność zapisu danych sprawdzana jest poprzez zbadanie, czy funkcja `fwrite()` zwróciła taką samą liczbę, jaka jej została przekazana jako trzeci argument wywołania. Ta liczba określa ile porcji danych będzie jednorazowo zapisywanych do pliku i w opisywanym przypadku jest to jedna porcja. Przed rozpoczęciem zapisu plik jest otwierany, a po jego zakończeniu zamykany. Jeśli podczas wykonywania dowolnej z czynności związanych z obsługą pliku pojawi się wyjątek, to funkcja przerwie swoje działanie i zwróci liczbę ujemną. W przypadku, gdy wszystkie operacje zakończą się sukcesem funkcja zwróci liczbę zero.

Przykład trzeci

```
int add_to_file(char *file_name)
{
    FILE *file = fopen(file_name,"a");
    if(file==NULL)
        return -4;
    struct coordinates point;
    const int RANGE = 40;
    point.x = rand()%RANGE;
    point.y = rand()%RANGE;
    point.z = rand()%RANGE;
    if(fwrite(&point,sizeof(point),1,file)!=1)
        return -5;
    if(fclose(file)!=0)
        return -3;
    return 0;
}
```

Przykład trzeci

Komentarz

Funkcja `add_to_file()` podobna jest w działaniu do funkcji opisanej poprzednio, ale występuje między nimi kilka znaczących różnic. Tym razem plik nie jest otwierany do zapisu, a do dopisywania, co oznacza, że jeśli plik istniał, to jego zawartość nie jest kasowana, a wskaźnik tego pliku ustawiany jest na jego końcu. Zapisywana jest tylko jedna struktura do pliku, a wartości dla jej pól losowane są z zakresu od 0 do 39. Kontrola i obsługa wyjątków, oraz pozostałe operacje zrealizowane są podobnie jak w funkcji `write_to_file()`.

Przykład trzeci

```
int read_from_file(char *file_name)
{
    FILE *file = fopen(file_name,"r");
    if(file==NULL)
        return -6;
    int i=0;
    struct coordinates point;
    while(fread(&point,sizeof(point),1,file)==1)
        printf("Punkt %d -- x: %f, y: %f z: %f\n",
                            ++i,point.x, point.y, point.z);
    if(fclose(file)!=0)
        return -3;
    return 0;
}
```

Przykład trzeci

Komentarz

Funkcja `read_from_file()` otwiera do odczytu plik binarny, którego nazwa została jej przekazana przez parametr, a następnie odczytuje w pętli wszystkie struktury z tego pliku i wypisuje zwartość ich pól na ekran w osobnych wierszach. Po zakończeniu odczytu funkcja zamyka plik. Wykrywanie końca pliku tym razem zostało zrealizowane inaczej niż w poprzednich programach. Badane jest co zwróciło wywołanie funkcji `fread()`. Dopóki zwraca ona informację, że udało jej się odczytać jedną (kolejną) strukturę, dotąd pętla jest wykonywana. W przeciwnym przypadku wykonanie pętli jest kończone. Obsługa wyjątków zrealizowana jest podobnie jak w poprzednio opisywanych funkcjach.

Przykład trzeci

```
int main(void)
{
    srand(time(0));
    int result = write_to_file("dane.bin");
    if(result!=0)
        display_exception_message(result);
    result = add_to_file("dane.bin");
    if(result!=0)
        display_exception_message(result);
    result = read_from_file("dane.bin");
    if(result!=0)
        display_exception_message(result);
    return 0;
}
```

Przykład trzeci

Komentarz

Funkcja `main()` zdefiniowana jest analogicznie jak w poprzednio opisywanych programach. Nowym elementem jest wywołanie funkcji, która dopisuje nowe dane (nową strukturę) na końcu istniejącego już pliku. W funkcji `main()` jest także inicjowany generator liczb pseudolosowych.

Przykład czwarty

Czwarty, ostatni przykładowy program zapisuje do pliku tekstowego wylosowane małe litery i liczby naturalne wierszami, po cztery wartości oddzielone od siebie spacjami. W programie zastosowano trochę inne podejście do obsługi wyjątków, które jest skuteczniejsze od zaprezentowanych poprzednio, ale też nie jest doskonałe, bo funkcja odczytująca plik nie jest informowana o tym, czy jego utworzenie się powiodło.

Przykład czwarty

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

#define LENGTH 50
```

Przykład czwarty

Komentarz

Do programu włączane są te same pliki nagłówkowe, co w poprzednim programie. Zdefiniowana stała o nazwie `LENGTH` określa liczbę elementów w tablicy znaków będącej parametrem, przez który do funkcji zapisujących i odczytujących dane przekazywana jest nazwa pliku.

Przykład czwarty

```
void write_to_file(char file_name[LENGTH])
{
    srand(time(0));
    FILE *file = fopen(file_name,"w");
    if(file!=NULL) {
        int i;
        for(i=0; i<5; i++) {
            fprintf(file,"%c %d %d %d\n",
                (char)('a'+rand()%('z'-'a'+1)),
                        rand()%5, rand()%7, rand()%3);
            if(ferror(file)) {
                perror("fprintf()");
                break;
            }
        }
        if(fclose(file))
            perror("fclose()");
    }
}
```

Przykład czwarty

Komentarz

Funkcja `write_to_file()`, po zainicjowaniu generatora liczb pseudolosowych, otwiera do zapisu plik tekstowy, którego nazwa jest jej przekazana przez parametr, a następnie zapisuje do niego pięć wierszy, z których każdy zawiera wylosowaną małą literę oraz trzy liczby naturalne losowane z różnych zakresów. Proszę zwrócić uwagę, że zapis nie jest wykonywany, jeśli nie udało się otworzyć pliku. Kontrola poprawności zapisu jest dokonywana poprzez wywoływanie funkcji `ferror()` po każdym wywołaniu funkcji `fprintf()`. Po zakończeniu zapisu, niezależnie od tego czy wystąpił wyjątek, czy nie plik jest zamykany, ale, ponownie, tylko wtedy gdy wcześniej udało się go otworzyć. Komunikaty o ewentualnych wyjątkach są wypisywane na ekranie monitora z użyciem funkcji `perror()`.

Przykład czwarty

```
void read_from_file(char file_name[LENGTH])
{
    char letter;
    int first_number, second_number, third_number;
    FILE *file = fopen(file_name,"r");
    if(file!=NULL) {
        while(!feof(file)) {
            fscanf(file,"%c %d %d %d\n",&letter,&first_number,
                                                &second_number,&third_number);
            if(ferror(file)) {
                perror("fscanf()");
                break;
            }
            printf("%c %d %d %d\n",letter,first_number,
                                                second_number,third_number);
        }
        if(fclose(file))
            perror("fclose()");
    }
}
```

Przykład czwarty

Komentarz

Funkcja `read_from_file()` zgodnie ze swoją nazwą odczytuje zwartość pliku, którego nazwa została jej przekazana przez parametr i wypisuje ją na ekran. Plik jest najpierw otwierany do odczytu, następnie w pętli dopóki są w nim dane jest odczytywany wiersz po wierszu za pomocą funkcji `fscanf()`. Proszę zwrócić uwagę, że ciąg formatujący jest identyczny jak dla funkcji `fprintf()`, która zapisywała dane do pliku. Również obsługa wyjątków jest zorganizowana podobnie jak w funkcji opisywanej poprzednio.

Przykład czwarty

```
int main(void)
{
    write_to_file("liczby.txt");
    read_from_file("liczby.txt");
    return 0;
}
```

Przykład czwarty

Komentarz

Funkcja `main()` w opisywanym programie jest dosyć prosta. Oprócz typowych elementów zawiera jedynie wywołania dwóch opisywanych wcześniej funkcji. Kody ich wykonania nie jest badany, ponieważ one nic nie zwracają.

Podziękowania

Składam podziękowania dla dra inż. Grzegorza Łukawskiego i mgra inż. Leszka Ciopińskiego za udostępnienie materiałów, których fragmenty zostały wykorzystane w tym wykładzie.

Pytania

?

KONIEC

Dziękuję Państwu za uwagę.