Podstawy Programowania 1

Obsługa terminala - biblioteka *curses*

Arkadiusz Chrobot

Zakład Informatyki

27 stycznia 2020

Plan

Wprowadzenie

Język C został pierwotnie zaprojektowany i stworzony na potrzeby systemu Unix, który początkowo komunikował się z użytkownikami za pomocą terminali złożonych z klawiatury i monitora. Monitor w terminalu pracował w trybie tekstowym, czyli takim, który pozwala na wyświetlanie wyłącznie znaków. Rozdzielczość ekranu takiego monitora mierzy się liczbą znaków, które może on jednocześnie wyświetlić. Najpopularniejszą rozdzielczością dla trybu tekstowego jest 80×25, czyli 80 kolumn i 25 wierszy. Niektóre z trybów tekstowych pozwalają również zastosować kolory dla wyświetlanych znaków oraz ich tła. Współczesne komputery wyświetlają obraz w trybie graficznym, jednakże tryb tekstowy nadal jest przez nie udostępniany. W pewnych zastosowaniach umożliwia on użytkownikowi wydajniejszą pracę niż tryb graficzny. Zauważono jednakże, że niektóre udogodnienia oferowane przez aplikacje pracujące w trybie graficznym można przenieść także do trybu tekstowego. W ten sposób powstała biblioteka o niezbyt wdzięcznej nazwie - *curses*.

Biblioteka *curses*

Biblioteka *curses* istnieje co najmniej w trzech odmianach. Dla systemów kompatybilnych z Uniksem, takich jak Linux dostępna jest biblioteka *ncurses* (od *new curses*). Dla systemów rodziny MS Windows opracowano bibliotekę *pdcurses* (od *public domain curses*). Oryginalna biblioteka *curses* dostępna jest dla systemów Unix. Nowe wersje tej biblioteki umożliwiają obsługę myszy oraz używanie elementów interfejsu użytkownika (ang. *widgets*) podobnych do tych jakie znane są z aplikacji pracujących w trybie graficznym. Dostępny jest nawet cały zestaw takich elementów nazwany CDK (ang. *Curses Development Kit*). Na tym wykładzie zostaną przedstawione tylko podstawowe możliwości oferowane przez *curses* i pokrewne biblioteki. Osoby bardziej zainteresowane tematem mogą znaleźć wiele innych źródeł informacji, np. stronę o tym adresie http://tldp.org/HOWTO/NCURSES-Programming-HOWTO/.

Inicjacja i sprzątanie

Aby użyć w programie biblioteki *curses* należy załączyć do niego plik nagłówkowy `curses.h`. Podstawową funkcją inicjującą pracę biblioteki *curses* jest `initscr()`. Nie przyjmuje ona żadnych argumentów wywołania, ale zwraca wskaźnik na strukturę typu `WINDOW`. Najczęściej ta wartość jest ignorowana, ale warto zbadać, czy nie jest ona równa `NULL`. Jeśli tak będzie, to oznacza to, że nie udało się zainicjować działania biblioteki. Inne funkcje, związane z inicjacją biblioteki są wymienione w tabeli na następnym slajdzie.

Inicjacja i sprzątanie

`echo()/noecho()`	Obie funkcje nie przyjmują żadnych argumentów wywołania i zwracają wartość typu `int`. Pierwsza powoduje, ze znaki pisane na klawiaturze pojawiają się na ekranie, druga ma działanie odwrotne.
`keypad()`	Funkcja inicjuje obsługę dodatkowych klawiszy na klawiaturze, takich jak klawisze kursorów, klawisze funkcyjne itd. Zwraca ona wartość typu `int`, a przyjmuje dwa argumenty wywołania, wskaźnik na strukturę okna (będzie objaśniona później) oraz wartość typu `bool`, która włącza/wyłącza obsługę wymienionych klawiszy.
`halfdelay()`	Ta funkcja włącza tryb obsługi klawiatury, w którym odpowiednie funkcje mogą odczytywać znaki z klawiatury pojedynczo i w ciągu ustalonego odcinka czasu. Funkcja przyjmuje jeden argument wywołania typ `int` oraz zwraca wartość tego samego typu. Wartość argumentu to liczba dziesiętnych sekund, przez które funkcje będą oczekiwały na naciśnięcie klawisza przez użytkownika.
`curs_set()`	Funkcja określająca sposób prezentacji kursora. Zwraca wartość typu `int` oraz przyjmuje jeden argument tego samego typu. Ten argument może być jedną z trzech liczb `0` - kursor jest niewidoczny, `1` - kursor ma kształt znaku podkreślenia, `2` - kursor ma postać bloku.

Wszystkie wymienione w tabeli funkcje zwracają wartość określoną stałą `OK`, jeśli ich wywołanie zakończyło się sukcesem lub `ERR` jeśli niepowodzeniem.

Inicjacja i sprzątanie

Za wyłączenie biblioteki odpowiada funkcja `endwin()` nie przyjmuje ona żadnych argumentów wywołania, ale zwraca wartość określoną stałą `OK` jeśli jej działanie powiedzie się lub `ERR` w przeciwnym przypadku.

Okno główne

Po zainicjowaniu pracy biblioteki *curses* za pomocą funkcji `initscr()` staje się dostępna zmienna `stdscr`. Jest to wskaźnik na strukturę `WINDOW`, która związana jest z ekranem głównym. Typ tej struktury jest zdefiniowany w bibliotece. Zmienne tego typu opisują atrybuty pojedynczego okna. Zmienna `stdscr` wskazuje na strukturę, która opisuje stan okna głównego, obejmującego cały ekran. Istnieją funkcje, które pozwalają umieszczać na tym ekranie znaki, sterować kursorem i wykonywać inne czynności. Punktem początkowym tego ekranu jest lewy górny róg, który ma współrzędne $(0, 0)$. Współrzędne pionowe (wierszy) rosną „w dół". Liczba dostępnych wierszy jest określona stałą `LINES`. Współrzędne poziome rosną „z lewa na prawo". Liczba dostępnych kolumn jest określona stałą `COLS`.

Nowe okna

Funkcja `newwin()` służy do tworzenia nowych okien, których obszar powinien być mniejszy lub równy rozmiarowi okna głównego, a położenie nie powinno wykraczać poza to okno. Tak utworzone okna nie mogą nakładać się na siebie. Jeśli potrzebujemy utworzyć nakładające się okna, to powinniśmy skorzystać z dodatkowej biblioteki o nazwie *panel*, która nie będzie przedstawiona na tym wykładzie. Funkcja `newwin()` przyjmuje cztery argumenty. Pierwszy to liczba wierszy nowego okna, drugi to liczba kolumn, a trzeci i czwarty argument to współrzędne miejsca w oknie głównym, gdzie będzie umieszczony lewy górny róg nowego okna. **Proszę zwrócić uwagę, na to, że wszystkie funkcje w bibliotece *curses* najpierw przyjmują jako argumenty wywołania składową pionową współrzędnych, a dopiero potem składową poziomą.** Jeśli uda się utworzyć nowe okno, to funkcja `newwin()` zwraca wskaźnik na strukturę `WINDOW` opisującą je.

Usuwanie okna

Okno główne jest usuwane po wywołaniu opisanej wcześniej funkcji `endwin()`, natomiast okna utworzone za pomocą `newwin()` usuwane są za pomocą wywołania funkcji `delwin()`. Przyjmuje ona jako argument wywołania wskaźnik typu `WINDOW`, a zwraca wartość typ `int`, która jest interpretowana tak samo, jak w przypadku innych funkcji z biblioteki *curses*.

Obsługa okna

Za pomocą funkcji `mvwin()` można zmieniać położenie okna względem okna głównego. Ta funkcja przyjmuje trzy argumenty wywołania. Pierwszym jest wskaźnik na strukturę `WINDOW` okna, które ma zostać przesunięte. Drugim i trzecim są składowe (y, x) nowego położenia górnego lewego rogu okna. Do czyszczenia zawartości okien służą funkcje `erase()` i `werase()`. Pierwsza czyści okno główne, druga okna utworzone za pomocą `newwin()`, dlatego jako argument wywołania przyjmuje wskaźnik na strukturę opisującą pojedyncze takie okno. Okno w przypadku biblioteki *curses* jest po prostu fragmentem ekranu i jest niewidoczne. Aby je uwidocznić możemy obrysować je ramką. Tę operację można wykonać na kilka sposobów. Najprostszym jest użycie funkcji `box()`. Przyjmuje ona trzy argumenty wywołania. Pierwszym jest wskaźnik na strukturę opisującą okno, które ma być obrysowane, drugim jest znak, który będzie użyty do rysowania fragmentów poziomych ramki, a trzecim znak, który będzie użyty do rysowania fragmentów pionowych ramki. Te znaki są określane przez stałe zaczynające się od od przedrostka `ACS_`. Jeśli podstawimy za nie wartości zero, to funkcja użyje znaków domyślnych.

Odświeżanie okien

Wszystkie operacje wykonywane na oknach są pierwotnie przeprowadzane we fragmentach pamięci operacyjnej, które im odpowiadają. Te fragmenty nazywane są *oknami wirtualnymi*. Aby zmiany wywołane przez funkcje stały się widoczne, należy zawartość okien, których one dotyczyły, odświeżyć, czyli przenieść zawartość okna wirtualnego do okna fizycznego. Funkcja `refresh()` realizuje tę operację dla okna głównego. Funkcja `wrefresh()` aktualizuje z kolei okno, którego struktura została jej przekazana przez wskaźnik jako argument wywołania. Jeśli zachodzi potrzeba aktualizacji większej liczby okien, to lepiej to zrobić wywołując najpierw dla każdego z nich funkcję `wnoutrefresh()`, a następnie tylko raz funkcję `doupdate()`. Pierwsza funkcja przyjmuje jako argument wywołania wskaźnik na strukturę opisującą aktualizowane okno, a druga nie przyjmuje żadnych argumentów wywołania. Wszystkie opisane tu i na poprzednim slajdzie funkcje zwracają wartości typu `int`, które w przypadku pomyślnego ich zakończenia mają wartość stałej `OK`, a w przypadku niepowodzenia działania `ERR`.

Położenie kursora

Funkcja `move()` służy do przeniesienia kursora w określone miejsce w oknie głównym. Jako argumenty wywołania przyjmuje składowe współrzędnych tego miejsca. Funkcja `wmove()` robi to samo, ale dla okna stworzonego przez `newwin()`. Przyjmuje ona trzy argumenty, pierwszym jest wskaźnik na strukturę opisującą okno, w którym kursor jest przemieszczany, a pozostałymi są składowe współrzędnych nowego położenia kursora. Te współrzędne liczone są względem lewego górnego rogu okna. Obie opisane funkcje zwracają wartość typu `int`, która interpretowana jest tak samo, jak w przypadku funkcji przedstawionych na dwóch poprzednich slajdach. Aby poznać bieżące współrzędne kursora możemy użyć makra o nazwie `getyx`. Przyjmuje ono trzy argumenty: zmienną wskaźnikową typu `WINDOW *`, oraz dwie zmienne typu `int`. Jeśli odczyt współrzędnych kursora się nie powiedzie, to makro umieści w dwóch ostatnich zmiennych wartość `-1`.

Wyświetlanie tekstu

Biblioteka *curses* dostarcza kilku funkcji, które pozwalają na umieszczenie znaków lub ciągu znaków na ekranie. Oprócz tego, że wykonują one podobne zadania, jak ich odpowiedniczki zadeklarowane w pliku nagłówkowym `stdio.h`, to niektóre z tych funkcji pozwalają nadać wyświetlanemu tekstowi odpowiednie atrybuty, takie jak pogrubienie lub podkreślenie. Wszystkie funkcje biblioteki *curses*, które są związane z wyświetlaniem tekstu zwracają wartości typu `int`. W przypadku, kiedy wypisanie zakończy się powodzeniem zwracają one wartość stałej `OK`, a w przeciwnym przypadku stałej `ERR`.

Wyświetlanie pojedynczych znaków

W bibliotece *curses* zdefiniowano szereg funkcji, których działanie odpowiada zachowaniu funkcji `putchar()` ze standardowej biblioteki języka C. Najprostszymi są `addch()` oraz `waddch()`. Pierwsza przyjmuje tylko jeden argument wywołania, który jest znakiem do wyświetlenia (lub stałą lub zmienną określającą taki znak). Znak ten wyświetlany będzie po aktualizacji w oknie głównym, w miejscu gdzie znajduje się kursor. Druga działa podobnie, ale jako pierwszy argument przyjmuje wskaźnik na strukturę określającą okno, gdzie znak ma się ukazać (`WINDOW`). Obie funkcje pozwalają nadać znakowi atrybut poprzez dodanie do argumentu będącego znakiem, stałej określającej ten atrybut, za pomocą operatora sumy bitowej - |. Przykładem takich stałych są `A_UNDERLINE` - podkreślenie, `A_BOLD` - pogrubienie. Więcej takich stałych można znaleźć w dokumentacji znajdującej się pod adresem podanym na początku wykładów.

Wyświetlanie pojedynczych znaków

Podobnie do opisanych na poprzednim slajdzie funkcji działają `mvaddch()` i `mvwaddch()`, ale przyjmują one dwa dodatkowe argumenty. W przypadku funkcji `mvaddch()` pierwsze dwa argumenty to, odpowiednio, składowa pionowa i pozioma miejsca na ekranie, gdzie znak zostanie wyświetlony. Te same dane przyjmuje funkcja `mvwaddch()`, ale jako drugi i trzeci argument wywołania. Jej pierwszym argumentem jest wskaźnik na strukturę okna, w którym znak ma być umieszczony. Wszystkie funkcje wyświetlające pojedyncze znaki mogą wypisywać na ekran także znaki specjalne, które określone są wartościami stałych, których nazwy rozpoczynają się przedrostkiem `ACS_`, np. `ACS_BULLET`, `ACS_LARROW`. Te stałe obsługiwane są również przez funkcje opisane na następnym slajdzie. Opis większej liczby stałych tego typu można znaleźć na stronie, której adres został podany na początku wykładu.

Wyświetlanie ciągów znaków

Odpowiednikami funkcji `puts()` ze standardowej biblioteki języka C są funkcje wymienione w tabeli.

`addstr()`	Funkcja przyjmuje jeden argument wywołania. Jest nim ciąg znaków, który zostanie po aktualizacji okna głównego wyświetlony na ekranie w miejscu, gdzie bieżąco znajduje się kursor.
`addnstr()`	Działa ona podobnie jak funkcja opisana wyżej, ale przyjmuje drugi argument, którym jest maksymalna liczba znaków, jakie mogą się znajdować w ciągu.
`waddstr()`	Ta funkcja działa jak `addstr()`, ale jako pierwszy argument przyjmuje wskaźnik na strukturę opisującą okno, w którym ciąg ma być wyświetlony.
`waddnstr()`	Funkcja działa jak ta opisana wyżej, ale przyjmuje trzeci argument określający maksymalną dopuszczalną liczbę znaków w wypisywanym ciągu.
`mvaddstr()`	Funkcja działa jak `addstr()`, ale jako pierwszy i drugi argument przyjmuje składową pionową i poziomą miejsca na ekranie, od którego ma zacząć wypisywać ciąg.
`mvaddnstr()`	Funkcja działa jak ta, która opisana jest wyżej, ale przyjmuje czwarty argument, którym jest maksymalna dopuszczalna liczba znaków w ciągu.
`mvwaddstr()`	Działa ona jak `mvaddstr()`, ale jako pierwszy argument przyjmuje wskaźnik na strukturę opisującą okno, gdzie będzie wyświetlony ciąg.
`mvwaddnstr()`	Działa ona jak funkcja opisana wyżej, ale jako ostatni argument przyjmuje maksymalną liczbę znaków w ciągu.

Wyświetlanie ciągów znaków

Biblioteka *curses* dostarcza również odpowiedników funkcji `printf()`:

`printw()`	Funkcja ta przyjmuje te same argumenty wywołania, co `printf()` i wyświetla ciąg znaków w oknie głównym, po jego odświeżeniu, począwszy od bieżącej pozycji kursora.
`wprintw()`	Działa ona jak funkcja opisana wyżej, ale jako pierwszy argument wywołania przyjmuje wskaźnik na strukturę okna, w którym ma być wyświetlony ciąg.
`mvprintw()`	Funkcja działa podobnie jak `printw()`, ale jako pierwszy i drugi argument wywołania przyjmuje odpowiednio składową pionową i składową poziomą współrzędnych punktu w oknie głównym, od którego ma zacząć wypisywanie ciągu.
`mvwprintw()`	Ta funkcja działa jak opisana wyżej `mvprintw()`, ale jako pierwszy argument przyjmuje wskaźnik na strukturę opisującą okno, w którym ma być wyświetlony napis. Pozostałe argumenty są takie, jak w przypadku `mvprintw()`.

Atrybuty znaków

Sposób nadawania atrybutów znakom wypisywanym przez funkcję `addch()` i pokrewne był opisany w części wykładu im poświęconej. Biblioteka *curses* pozwala również nadawać atrybuty ciągom znaków wypisywanych przez pozostałe z opisanych funkcji. W przypadku funkcji `addstr()` i pokrewnych możemy zastosować to samo rozwiązanie, co dla pojedynczych znaków. W przypadku funkcji `printw()` i pokrewnych musimy zastosować odpowiednie funkcje zarządzające atrybutami. Dla okna głównego będą to funkcje `attron()`, `attrset()` oraz `attroff()`. Wszystkie one pobierają jeden argument wywołania typu `int`, którym najczęściej jest stała określająca atrybut. Również wszystkie te funkcje zwracają wartość typu `int`, która odpowiada stałej `OK` lub `ERR`. Funkcja `attron()` włącza atrybut. Jeśli zostanie ona wywołana kilka razy pod rząd, z różnymi atrybutami jako argumentami jej wywołania, to wszystkie one zostaną włączone. Funkcja `attrset()`, jeśli zostanie ponownie wywołana z innym argumentem niż poprzednio, to wyłącza poprzednio ustawiony przez siebie argument i włącza ten, z którym ostatnio była wywołana. Funkcja `attroff()` po prostu wyłącza przekazany jej jako argument atrybut.

Atrybuty znaków

Istnieją odpowiedniki funkcji opisanych na poprzednim slajdzie, które obsługują dowolne okna. Ich nazwy zostały utworzone od nazw wspomnianych funkcji poprzez dodanie na ich początku litery `w`. Przyjmują one dwa parametry, pierwszym jest wskaźnik na strukturę okna, w którym operacja zmiany atrybutów tekstu ma być przeprowadzona, a drugim wartość opisująca atrybut.

Obsługa klawiatury

Biblioteka *curses* dostarcza również funkcji pozwalających sterować klawiaturą i odczytywać z niej zarówno pojedyncze znaki, jak i całe ciągi znaków. Wypisanie znaków odczytywanych przez te funkcje na ekranie nie wymaga aktualizacji jego zawartości. Zachowanie tych funkcji zależy od sposobu inicjacji pracy biblioteki w programie. Przykładem funkcji mających wpływ na to zachowanie są opisane wcześniej `echo()` i `noecho()`.

Odczyt pojedynczych znaków

Niektóre wersje biblioteki od razu włączają tryb, w którym odczytywane są naciśnięcia pojedynczych klawiszy, inne zachowują tryb znany ze standardowej biblioteki wejścia-wyjścia - każdy znak musi być potwierdzony klawiszem `Enter`. W tym drugim przypadku możemy przejść do trybu odczytu pojedynczych klawiszy np. za pomocą wywołania funkcji `cbreak()`. Nie pobiera ona żadnych argumentów wywołania, a zwracana przez nią wartość jest interpretowana, tak jak w przypadku wszystkich wcześniej opisanych funkcji biblioteki *curses*, które zwracają wartość typu `int`. Standardowy tryb obsługi klawiatury przywraca również bezargumentowa funkcja `nocbrake()`, zwracająca wartość typu `int`. To, czy funkcje odczytujące znaki mają działać blokująco, tzn. czekać aż użytkownik wprowadzi dane, lub nieblokująco można określić za pomocą wywołania funkcji `nodelay()`, która przyjmuje dwa argumenty: wskaźnik na strukturę okna, dla którego będzie wykonana ta operacja i wartość typu `bool`, która określa, czy włączyć działanie nieblokujące, czy też nie. Funkcja ta zwraca wartość typu `int`. Inną funkcją, która ma wpływ na zachowanie funkcji odczytujących z klawiatury pojedyncze znaki jest opisana wcześniej `halfdelay()`.

Odczyt pojedynczych znaków

Pojedyncze znaki z klawiatury mogą być odczytane za pomocą bezargumentowej funkcji `getch()`, która zwraca kody ASCII odczytanych znaków, jako wartości typu `int`. Jeśli wyświetlanie znaków odczytanych z klawiatury (echo) jest włączone, to taki znak pojawi się w oknie głównym, w miejscu gdzie bieżąco znajduje się kursor. Jeśli włączony jest nieblokujący odczyt znaków z klawiatury, to jeśli użytkownik nie naciśnie żadnego klawisza, to funkcja ta zwróci wartość określoną stałą `ERR`. Odpowiedniczką `getch()`, pozwalającą wyświetlić odczytywane znaki w dowolnym oknie jest funkcja `wgetch()`. Przyjmuje ona jako argument wskaźnik na strukturę opisującą okno. Funkcje `mvgetch()` i `mvwgetch()` pozwalają określić miejsce na ekranie, gdzie odczytany znak zostanie wypisany. Pierwsza przyjmuje dwa argumenty typu `int`. Pierwszy jest pionową, a drugi poziomą składową współrzędnych miejsca w oknie główny, gdzie zostanie wypisany znak. Druga funkcja jako pierwszy argument przyjmuje wskaźnik na strukturę okna, a pozostałe argumenty są takie same, jak dla `mvgetch()`.

Odczyt pojedynczych znaków

Biblioteka *curses* definiuje szereg stałych, których wartościami są kody naciśniętych klawiszy specjalnych. Przykładowo, stała odpowiadająca klawiszowi przesuwającemu kursor na ekranie w lewo ma nazwę `KEY_LEFT`. Istnieje także makro o nazwie `KEY_F`, które zwraca kod klawiszy funkcyjnych (`F1`, `F2`, itd.). Jako argument rozwinięcia przyjmuje ono numer takiego klawisza. Użycie tych stałych ma uzasadnienie, jeśli została wcześniej w programie włączona obsługa dodatkowych klawiszy klawiatury. Więcej informacji o tych stałych można znaleźć na stronie, której adres został podany na początku wykładu.

Odczyt ciągów znaków

Istnieją w bibliotece *curses* funkcje odczytujące ciągi znaków, które działają analogicznie jak `fgets()` ze standardowej biblioteki wejścia-wyjścia. Zostały one opisane w tabeli.

`getstr()`	Funkcja wczytuje ciąg znaków do tablicy przekazanej jej jako argument wywołania. **Jej działanie jest niebezpieczne, bo nie ogranicza liczby wczytywanych znaków. Lepiej jej nie używać.**
`getnstr()`	Funkcja działa tak jak ta opisana wyżej, ale przyjmuje dodatkowy argument, jakim jest maksymalna liczba znaków, jakie może odczytać z klawiatury, dlatego jej użycie jest bezpieczniejsze.
`wgetstr()`	Odpowiedniczka `getstr()` działająca dla wszystkich okien. Jako pierwszy argument przyjmuje wskaźnik na strukturę opisującą okno, gdzie ma być wypisany odczytany z klawiatury ciąg znaków. **Również lepiej jej nie używać.**
`wgetnstr()`	Bezpieczniejsza wersja `wgetstr()`. Jako ostatni argument przyjmuje maksymalną liczbę znaków, które mogą być wczytane.
`mvgetstr()`	Odpowiedniczka `getstr()`, przyjmuje jako dwa pierwsze argumenty składową pionową i poziomą współrzędnych miejsca w oknie głównym, od którego ma rozpocząć się wypisywanie znaków odczytanych z klawiatury. **Należy unikać jej używania.**
`mvwgetstr()`	Odpowiedniczka `mvgetstr()`, która jako pierwszy argument przyjmuje wskaźnik na strukturę opisującą okno, gdzie ma być wypisany odczytany z klawiatury ciąg znaków. **Również lepiej unikać jej stosowania.**

Odczyt ciągów znaków

`mvgetnstr()`	Bezpieczniejsza wersja `mvgetstr()`. Maksymalna liczba wczytywanych znaków jest podawana jako ostatni argument jej wywołania.
`mvwgetnstr()`	Bezpieczniejsza wersja `mvwgetstr()`. Maksymalna liczba wczytywanych znaków jest podawana jako ostatni argument jej wywołania.

Wszystkie opisane w tabeli funkcje zwracają wartość typ `int`. Jeśli jest ona równa wartości stałej `OK`, to działanie funkcji zakończyło się powodzeniem, w przeciwnym wypadku zwracana wartość jest równa wartości stałej `ERR`.

Odczyt ciągów znaków

Biblioteka *curses* dostarcza również odpowiedniczek funkcji `scanf()`:

`scanw()`	Funkcja ta przyjmuje takie same argumenty jak `scanf()`. Oczytany z klawiatury ciąg znaków jest wyświetlany w głównym oknie.
`wscanw()`	Funkcja ta działa podobnie do `scanw()`, ale jako pierwszy argumenty przyjmuje wskaźnik typu `WINDOW *`, do struktury opisującej okno, gdzie zostanie wypisany ciąg znaków odczytany z klawiatury.
`mvscanw()`	Funkcja działa podobnie jak `scanw()`, ale jako pierwsze dwa argumenty przyjmuje składową pionową i poziomą miejsca w oknie głównym, od którego rozpocznie wypisywanie odczytanego z klawiatury ciągu znaków.
`mvwscanw()`	Funkcja działa podobnie jak `mvscanw()`, ale jako pierwszy argument przyjmuje wskaźnik na strukturę opisującą okno, w którym zostanie wypisany odczytany z klawiatury ciąg znaków.

Wszystkie funkcje z tabeli zwracają wartość typu `int`. Jeśli jest ona równa stałej `OK`, to ich działanie zakończyło się powodzeniem. W przeciwnym przypadku zwracają one wartość odpowiadającą stałej `ERR`.

Obsługa kolorów

Kolory są atrybutami wyświetlanych znaków. Nie wszystkie terminale pozwalają na ich stosowanie. Aby sprawdzić, czy dany terminal oferuje taką możliwość, należy w programie wywołać bezargumentową funkcję `has_colors()`. Zwraca ona wartość typu `bool`. Jeśli będzie ona równa `TRUE`, to terminal, z którym pracuje program, pozwala na stosowanie kolorów. Po sprawdzeniu dostępności kolorów należy wywołać bezargumentową funkcje `start_color()`, która włącza ich obsługę. Jeśli zwróci ona wartość odpowiadającą stałej `OK`, to będzie to znaczyło, że obsługa kolorów została włączona. W przeciwnym przypadku funkcja zwraca wartość odpowiadającą stałej `ERR`. Liczba dostępnych kolorów jest określona stałą `COLORS`. Biblioteka *curses* nie pozwala jednak używać kolorów pojedynczo. Należy wcześniej skonfigurować pary kolorów. Pierwszy kolor w parze nadawany jest znakowi, a drugi jego tłu. Maksymalną liczbę takich par określa stała `COLOR_PAIRS`. Do konfiguracji par służy funkcja `init_pair()`. Przyjmuje ona trzy argumenty wywołania typu `short`. Pierwszy określa numer pary, który musi być większy od zera, dwa pozostałe odpowiednio numer koloru znaku i numer koloru tła.

Obsługa kolorów

Funkcja `init_pair()` zwraca wartość typu `int` interpretowaną tak samo, jak w przypadku `start_color()`. Do wybierania pary kolorów służy makro `COLOR_PAIR`, które jako argument rozwinięcia przyjmuje numer pary kolorów i zwraca kolory opisywane przez tę parę jako atrybut znaków, który może być np. nadany za pomocą wywołania funkcji `attron()`. Tabela zawiera wykaz stałych opisujący kolory wraz z ich nazwą.

COLOR_BLACK	kolor czarny
COLOR_RED	kolor czerwony
COLOR_GREEN	kolor zielony
COLOR_YELLOW	kolor żółty
COLOR_BLUE	kolor niebieski
COLOR_MAGENTA	kolor karmazynowy
COLOR_CYAN	kolor turkusowy
COLOR_WHITE	kolor biały

Przykłady

Kody źródłowe wszystkich zaprezentowanych programów można znaleźć na stronie przedmiotu w wersji gotowej do użycia w środowisku Code::Blocks. Wszystkie, oprócz ostatniego zostaną zaprezentowane w całości. Ze względu na czytelność, w programach tych zastosowano bardzo prostą kontrolę wyjątków - jeśli działanie danej funkcji nie zakończy się poprawnie, to najczęściej powoduje to natychmiastowe zakończenie działania programu. Prawidłowa obsługa wyjątków powinna przed zakończeniem programu posprzątać po funkcjach inicjujących, których działanie się powiodło i przywrócić ustawienia terminala, które on posiadał przed uruchomieniem programu. Kody źródłowe ze strony przedmiotu udostępnione są wraz z plikami konfiguracyjnymi dla środowiska Code::Blocks dostosowanymi do użycia z biblioteką `ncurses`. Aby użyć ich z biblioteką `pdcurses` konieczne będzie wprowadzenie odpowiednich zmian.

Przykład 1 - prosty program

```
#include<curses.h>
#include<locale.h>

int main(void)
{
    if(setlocale(LC_ALL,"")==NULL)
        return -1;
    if(initscr()==NULL)
        return -1;
    printw("Witaj świecie!\n");
    if(refresh()==ERR)
        return -1;
    getch();
    if(endwin()==ERR)
        return -1;
    return 0;
}
```

Komentarz do przykładu 1

Działanie tego programu jest bardzo proste - inicjuje on pracę biblioteki, wyświetla na ekranie słynny napis „Witaj świecie” i czeka aż użytkownik naciśnie dowolny klawisz na klawiaturze, po czym wyłącza działanie biblioteki *curses* i kończy swoje wykonanie. Aby napis przekazany do wywołania funkcji `printw()` ukazał się na ekranie niezbędna jest aktualizacja okna głównego, za pomocą wywołania funkcji `refresh()`. Oczekiwanie na naciśnięcie przez użytkownika klawisza zrealizowane jest za pomocą wywołania funkcji `getch()`. Użyta na początku funkcja `setlocale()` jest zadeklarowana w pliku nagłówkowy `locale.h` i służy do ustawienia lokalizacji programu, aby napis wyświetlany przez program zawierał polskie litery.

Przykład 2 - odczyt klawiszy

```c
#include<ncurses.h>
#include<locale.h>

void print_keys(void)
{
    int key;
    do {
        key = getch();
        printw("Został naciśnięty klawisz %c\n",key);
        refresh();
    } while(key!='q');
}
```

Przykład 2 - odczyt klawiszy

```
int main(void)
{
    if(setlocale(LC_ALL,"")==NULL)
        return -1;
    if(initscr()==NULL)
        return -1;
    if(noecho()==ERR)
        return -1;
    print_keys();
    if(endwin()==ERR)
        return -1;
    return 0;
}
```

Przykład 2 - odczyt klawiszy

W funkcji `main()` wykonywana jest inicjacja pracy biblioteki *curses* i zmiana ustawień terminala. Dzięki wywołaniu funkcji `noecho()` funkcja `getch()` nie będzie drukowała na ekranie odczytanych znaków. Jest ona wywoływana w funkcji `print_keys()`, wewnątrz pętli `do...while`, która jest wykonywana tak długo, aż użytkownik naciśnie klawisz `q`. Informacje o znakach związanych z naciśniętymi klawiszami wypisuje na ekranie funkcja `printw()`. Proszę zwrócić uwagę, że naciśnięcie niektórych klawiszy, takich jak np. klawisze kursora, powoduje wypisanie na ekranie więcej niż jednego znaku.

Przykład 3 - poruszanie kursora

```
#include<ncurses.h>

void move_cursor(WINDOW *window)
{
    int x=0,y=0;
    getyx(window,y,x);
    int key = 0;
    do {
        key = getch();
        switch(key) {
        case KEY_LEFT:
            x=(x+(COLS-1))%COLS;
            move(y,x);
            break;
        case KEY_RIGHT:
            x=(x+1)%COLS;
            move(y,x);
```

Przykład 3 - poruszanie kursora

```
        break;
    case KEY_UP:
        y=(y+(LINES-1))%LINES;
        move(y,x);
        break;
    case KEY_DOWN:
        y=(y+1)%LINES;
        move(y,x);
        break;
    case KEY_F(3):
        getyx(window,y,x);
        printw("x: %d, y: %d",x,y);
        break;
```

Przykład 3 - poruszanie kursora

```
        case KEY_F(2):
            y=x=0;
            erase();
            break;
        }
        refresh();
    } while(key!='q');
}
```

Przykład 3 - poruszanie kursora

```
int main(void)
{
    if(initscr()==NULL)
        return -1;
    if(keypad(stdscr,TRUE)==ERR)
        return -1;
    move_cursor(stdscr);
    if(endwin()==ERR)
        return -1;
    return 0;
}
```

Przykład 3 - komentarz

Zaprezentowany program umożliwia poruszanie kursorem za pomocą klawiszy kursora po całym dostępnym ekranie. Inicjacja pracy biblioteki i sprzątanie po niej odbywa się w funkcji `main()` programu. Tam jest także wywoływana funkcja `keypad()`, która włącza obsługę klawiszy specjalnych przez funkcje obsługujące klawiaturę i działające w oknie głównym. Ruch kursora jest oprogramowany w funkcji `move_cursor()`. Argumentem wywołania tej funkcji jest wskaźnik do struktury opisującej okno główne. Wewnątrz funkcji `move_cursor()` wywoływane jest najpierw makro `getyx` celem początkowego ustalenia pozycji kursora na ekranie. Jego współrzędne są zapisywane w zmiennych `x` i `y`. Następnie deklarowana i inicjowana jest zmienna `key`, do której przy pomocy wywołań `getch()` w pętli `do...while` zapisywane są kody naciśniętych przez użytkownika klawiszy. Analiza tych kodów jest wykonywana w tej samej pętli, przy pomocy instrukcji `switch`. Jeśli ich wartość odpowiada którejś ze stałych `KEY_RIGHT`, `KEY_LEFT`, `KEY_UP` lub `KEY_DOWN`, to klawisz jest przesuwany o jedno miejsce na ekranie w odpowiednią stronę.

Przykład 3 - komentarz

Przesunięcie polega na wyliczeniu nowej wartości odpowiedniej składowej położenia kursora i wykonaniu funkcji `move()` dla nowych współrzędnych. Ponieważ zastosowano tu wyliczanie współrzędnych przy użyciu arytmetyki modularnej - rozwiązanie znane z programu gry w życie - to kursor po „dojściu" do krawędzi pojawi się z drugiej strony ekranu. Jeśli użytkownik naciśnie klawisz `F2`, to program wypisze na ekranie współrzędne poprzedniego położenia kursora, które zostaną odczytane za pomocą makra `getyx` i wypisane na ekranie z użyciem `printw()`. Jeśli użytkownik naciśnie klawisz `F3`, to program wyczyści okno główne wywołując funkcję `erase()` i wyzeruje zmienne pamiętające bieżące położenie kursora. W pętli wywoływana jest również funkcja `refresh()`, celem aktualizacji widoku okna głównego. Instrukcja iteracyjna kończy się po naciśnięciu przez użytkownika klawisza `q`.

Przykład 4 - okna

```
#include<ncurses.h>
#include<locale.h>

void move_window(WINDOW *window, int x, int y)
{
    int key=0;
    do {
        key = getch();
        if(key==' ') {
            x=(x+1)%10;
            y=(y+1)%10;
            erase();
            refresh();
            if(mvwin(window,y,x)==ERR)
                printw("Przesunięcie poza dozwolony obszar\n");
            if(wrefresh(window)==ERR)
                printw("Błąd odświeżania okna\n");
        }
    } while(key!='q');
}
```

Przykład 4 - okna

```
int main(void)
{
    if(setlocale(LC_ALL,"")==NULL)
        return -1;
    if(initscr()==NULL)
        return -1;
    if(curs_set(0)==ERR)
        return -1;
    WINDOW *window = newwin(5,10,0,0);
    if(window==NULL)
        return -1;
    if(box(window,0,0)==ERR)
        return -1;
    if(refresh()==ERR)
        return -1;
    if(wrefresh(window)==ERR)
        return -1;
    move_window(window,0,0);
    if(delwin(window)==ERR)
        return -1;
    if(endwin()==ERR)
        return -1;
    return 0;
}
```

Przykład 4 - komentarz

W funkcji `main()` najpierw wykonywana jest lokalizacja programu i inicjacja biblioteki *curses*. Następnie przy pomocy wywołania funkcji `curs_set()` wyłączany jest widok kursora - kursor staje się niewidoczny. Kolejnymi czynnościami wykonywanymi w ramach funkcji `main()` są utworzenie okna o wymiarach 5×10 w lewym górny rogu ekranu za pomocą funkcji `newwin()` i narysowanie ramki dla tego okna przy pomocy wywołania `box()`. Następnie aktualizowana jest zawartość zarówno okna głównego, jak i nowego okna. Potem wywoływana jest zdefiniowana w programie funkcja `move_window()`. Do niej przekazywany jest wskaźnik na strukturę opisującą nowo utworzone okno oraz początkowe współrzędne jego lewego górnego rogu. Wewnątrz tej funkcji, w pętli `do...while` odczytywany jest za pomocą `getch()` kod ASCII klawisza naciśniętego przez użytkownika i w przypadku, gdy jest to spacja, to wyliczana jest nowa pozycja lewego górnego rogu utworzonego w programie okna. Następnie okno główne jest czyszczone i wywoływana jest funkcja `mvwin()`, która przesuwa okno względem wyliczonych współrzędnych.

Przykład 4 - komentarz

Poprawność wykonania tej funkcji jest kontrolowana w programie, choć terminal musiałby mieć bardzo małą rozdzielczość, aby przesunięcie okna się nie powiodło. Przemieszczenie okna zostanie uwidocznione dopiero po wywołaniu funkcji aktualizującej jego zawartość, czyli `wrefresh()`. Stan jej wykonania również jest sprawdzany. Wykonanie pętli wewnątrz funkcji `move_window()` jest przerywane po naciśnięciu przez użytkownika klawisza `q`. Przed zakończeniem pracy biblioteki *curses* za pomocą wywołania `endwin()` wywoływana jest funkcja `delwin()`, która usuwa okno utworzone przez `newwin()`.

Przykład 5 - kolory

```
#include<curses.h>
#include<locale.h>

void init_color_pairs(void)
{
    short int i,j, pair_counter=1;
    for(i=COLOR_BLACK;i<COLOR_WHITE;i++)
        for(j=COLOR_BLACK;j<COLOR_WHITE;j++) {
            if(init_pair(pair_counter,i,j)==ERR) {
                    printw("Nie udało się zainicjować pary %d\n",pair_counter);
                    refresh();
            }
            pair_counter++;
        }
}
```

Przykład 5 - kolory

```
void test_colors(void)
{
    short int i;
    for(i=1; i<COLOR_PAIRS; i++) {
        attron(COLOR_PAIR(i));
        printw("Test pary kolorów nr %d\n",i);
        refresh();
        attroff(COLOR_PAIR(i));
        if(i%24==0) {
            getch();
            erase();
        }
    }
}
```

Przykład 5 - kolory

```
int main(void)
{
    if(setlocale(LC_ALL,"")==NULL)
        return -1;
    if(initscr()==NULL)
        return -1;
    if(curs_set(0)==ERR)
        return -1;
    if(!has_colors())
        return -1;
    if(start_color()==ERR)
        return -1;
    init_color_pairs();
    printw("Dostępne jest %d kolorów i %d par kolorów.\n",COLORS,COLOR_PAIRS);
    refresh();
    getch();
    erase();
    test_colors();
    getch();
    if(endwin()==ERR)
        return -1;
    return 0;
}
```

Przykład 5 - komentarz

W funkcji `main()` programu wykonywana jest lokalizacja, inicjowana jest praca biblioteki *curses* oraz wyłączany jest widok kursora. Następnie przy pomocy wywołania funkcji `has_colors()` sprawdzane jest, czy program pracuje z terminalem pozwalającym na używanie kolorów. Jeśli tak, to uruchamiana jest obsługa kolorów za pomocą wywołania funkcji `start_color()`, a potem tworzone są odpowiednie pary kolorów w zdefiniowanej w programie funkcji `init_color_pairs()`. Nie wszystkie z tych par są przydatne (np. czerwony znak na czerwonym tle nie będzie widoczny), ale w programie są prezentowane wszystkie możliwe. Poszczególne pary są generowane poprzez wywołanie `init_pair()` w podwójnej pętli `for`. Po zakończeniu funkcji `init_color_pairs()` wypisywana jest informacja o liczbie dostępnych kolorów i par kolorów, a następnie program przy pomocy funkcji `getch()` czeka aż użytkownik naciśnie dowolny klawisz. Kiedy to się stanie czyszczony jest ekran i wywoływana jest kolejna zdefiniowana w programie funkcja: `test_colors()`.

Przykład 5 - komentarz

Ta funkcja w pętli `for` nadaje tekstowi atrybut w postaci odpowiedniej pary kolorów za pomocą wywołania `attron()`, następnie wyświetla komunikat o takich kolorach za pomocą wywołania `printw()`, odświeża okno główne wywołując `refresh()` i wyłącza daną parę kolorów za pomocą `attroff()`. Wewnątrz pętli umieszczona jest także instrukcja warunkowa `if`, której celem jest sprawienie, aby program zatrzymał swoje wykonanie co 24 wypisane na ekranie wiersze, poczekał aż użytkownik naciśnie dowolny klawisz na klawiaturze, a następnie wyczyścił zawartość okna głównego. Po zakończeniu funkcji `test_colors()` program kończy pracę z biblioteką *curses* i kończy swoje działanie.

Przykład 6 - gra w życie

Kolejny przykład jest zmodyfikowaną wersją gry w życie, która była przedstawiona na wykładzie poświęconym tablicom wielowymiarowym. W tych materiałach zostaną przedstawione tylko te fragmenty programu, które uległy zmianie. Całość kodu źródłowego dostępna jest na stronie wykładu. Jedyną zmianą, która nie zostanie pokazana na następnych slajdach jest zamiana pliku nagłówkowego `stdio.h` na plik `curses.h`.

Przykład 6 - gra w życie

```
char *error_msg[] = {
    "OK",
    "initscr() error",
    "noecho() error",
    "halfdealy() error",
    "start_color() error",
    "init_pair() error",
    "curs_set() error",
    "endwin() error"
};
```

Przykład 6 - komentarz

Tablica `error_msg` zawiera ciągi znaków będące komunikatami opisującymi wyjątki jakie mogą w programie sygnalizować funkcje związane z biblioteką *curses*.

Przykład 6 - gra w życie

```
int initiate(void)
{
    if(!initscr())
        return -1;
    if(noecho()==ERR)
        return -2;
    if(halfdelay(2)==ERR)
        return -3;
    if(has_colors()!=FALSE) {
        if(start_color()==ERR)
            return -4;
        if(init_pair(1,COLOR_GREEN,COLOR_BLACK)==ERR ||
                init_pair(2,COLOR_BLACK,COLOR_BLACK)==ERR)
            return -5;
    }
    if(curs_set(0)==ERR)
        return -6;
    return 0;
}
```

Przykład 6 - komentarz

Funkcja `initiate()` odpowiedzialna jest za inicjację pracy biblioteki *curses*. W tej funkcji wyłączane jest także echo operacji związanych z odczytem klawiatury, sprawdzana jest dostępność kolorów, uruchamiana jest ich obsługa, definiowane są dwie pary kolorów (zielone znaki na czarnym tle i czarne znaki na czarnym tle) oraz wyłączana jest widoczność kursora. Ponadto terminal jest przełączany za pomocą wywołania `halfdelay()` do trybu, w którym program czeka na naciśnięcie przez użytkownika klawisza przez $0,2$ sekundy.

Przykład 6 - gra w życie

```
WINDOW *create_board_window(void)
{
    int middle_y = LINES/2;
    int middle_x = COLS/2;
    int half_board = SIZE/2;
    int start_x = middle_x - SIZE;
    int start_y = middle_y - half_board;
    return newwin(SIZE,2*SIZE,start_y,start_x);
}
```

Przykład 6 - komentarz

Funkcja `create_board_window()` tworzy okno na ekranie, wewnątrz którego będzie wyświetlany stan gry. Najpierw wyliczane są składowe współrzędnych środka ekranu, których wartości są zapamiętywane w zmiennych `middle_x` i `middle_y`. Następnie wyliczana jest połowa długości boku planszy i zapamiętywana w zmiennej `half_board`. Kolejną czynnością wykonywaną w funkcji jest wyliczenie współrzędnych lewego górnego rogu okna (zmienne `start_x` i `start_y`) i utworzenie tego okna. Proszę zauważyć, że szerokość tego okna będzie dwa razy większa od jego wysokości. Dzieje się tak, ponieważ kolumny są węższe od wierszy, a chcemy uzyskać na ekranie planszę kwadratową (a przynajmniej zbliżoną do kwadratu). Funkcja `create_board_window()` zwraca wskaźnik na nowo utworzone okno.

Przykład 6 - gra w życie

```
void print_board(unsigned char board[SIZE][SIZE], WINDOW *board_window)
{
    unsigned int i,j;

    for(i=0; i<SIZE; i++)
        for(j=0; j<SIZE; j++)
            if(board[i][j]) {
                (void)wattron(board_window,COLOR_PAIR(1));
                (void)mvwaddch(board_window,i,2*j,ACS_BULLET);
                (void)wattroff(board_window,COLOR_PAIR(1));
            } else {
                (void)wattron(board_window,COLOR_PAIR(2));
                (void)mvwaddch(board_window,i,2*j,' ');
                (void)wattroff(board_window,COLOR_PAIR(2));
            }
}
```

Przykład 6 - komentarz

Funkcja `print_board()` jest odpowiedzialna za wyświetlenie stanu gry, który zapisany jest w macierzy `board` na ekranie. Przez drugi parametr tej funkcji przekazywany jest wskaźnik do okna, w którym ten stan zostanie wyświetlony. Wewnątrz podwójnej pętli `for` badany jest każdy element macierzy. Jeśli jego wartość jest różna od zera, to jest on na ekranie odwzorowywany w postaci zielonej kropki (stała `ACS_BULLET`) na czarnym tle. Jeśli ta wartość wynosi zero, to jest on rysowany jako czarna spacja na czarnym tle. Nie będzie on wtedy widoczny, ale jego rysowanie jest konieczne, gdyż bieżący stan jest rysowany na poprzednim. W miejscu, gdzie teraz jest martwa komórka w poprzednim stanie mogła być żywa. W związku z tym trzeba będzie ją wymazać czarną spacją. Proszę zwrócić także uwagę na to, że ze względu na proporcje ekranu terminala wartości poziome współrzędnych punktów są mnożone przez dwa. Dzięki temu kształt widoku planszy jest kwadratowy. Umieszczenie słowa kluczowego `void` w nawiasach okrągłych przed wywołaniem funkcji oznacza, że wynik jej działania będzie ignorowany. Takie użycie `void` nie jest obowiązkowe i najczęściej jest pomijane. Brak przypisania wartości funkcji do zmiennej jest traktowany tak samo jak powyższy zapis.

Przykład 6 - gra w życie

```
int main(int argc, char **argv)
{
    int error = initiate();
    if(error!=0) {
        if(error<-1)
            (void)endwin();
        fprintf(stderr,"%s\n",error_msg[-error]);
        return -1;
    }
    WINDOW *board_window = create_board_window();
    if(!board_window) {
        printw("board_window() error\n");
        return -2;
    }
    if(argc==2) {
        if(!strcmp(argv[1],"blinker"))
            create_blinker(board);
        else if(!strcmp(argv[1],"ten_in_row"))
            create_ten_in_row(board);
        else
            seed_board(board);
    } else
        seed_board(board);
```

Przykład 6 - gra w życie

```
    while(getch()==ERR) {
        print_board(board,board_window);
        get_next_step(board);
        wrefresh(board_window);
    }

    if(delwin(board_window)==ERR) {
        printw("delwin() error\n");
        return -3;
    }

    if(endwin()==ERR) {
        fprintf(stderr,"%s\n",error_msg[7]);
        return -4;
    }

    return 0;
}
```

Przykład 6 - komentarz

W porównaniu z wersją zaprezentowaną na wykładzie o tablicach wielowymiarowych zmianie uległa również funkcja `main()`. Wywoływana jest w niej najpierw funkcja `initiate()`. Jeśli zwróci ona wartość różną od zera, to wypisywany jest komunikat o wyjątku inicjacji biblioteki *curses* i program skończy działanie. Dodatkowo, jeśli kod wyjątku będzie mniejszy od `-1`, co oznacza problem z działaniem funkcji wywoływanych po `initscr()`, to przed wypisaniem komunikatu wywoływana jest funkcja `endwin()` kończąca pracę biblioteki *curses*. Jeśli inicjacja przebiegnie prawidłowo, to tworzone jest nowe okno przy pomocy wywołania funkcji `create_board_window()`. Program również sprawdza kod jej wykonania, ale tym razem, w przypadku problemów, obsługa wyjątku jest pobieżna - kończone jest działanie programu, ale nie jest wykonywane sprzątanie po bibliotece. Zmianie uległa również pętla w funkcji `main()`. Oprócz tego, że wywoływana jest w niej nowa wersja funkcji `print_board()` i funkcja `wrefresh()`, to zmieniono również jej warunek zakończenia. Jest ona wykonywana tak długo, jak długo użytkownik nie naciśnie żadnego klawisza.

Przykład 6 - komentarz

Stan klawiatury w warunku pętli `while` jest badany za pomocą funkcji `getch()`, która zwraca wartość opisaną stałą `ERR` tak długo, jak długo użytkownik nie naciśnie żadnego klawisza na klawiaturze. Ponieważ w funkcji `initiate()` jest wywoływana `halfdelay()` z argumentem równym 2, więc funkcja `getch()` przy każdej iteracji pętli będzie czekała na naciśnięcie klawisza tylko przez `0,2` sekundy. Oznacza to, że program będzie wyświetlał kolejne stany gry co wspomniany odcinek czasu, a więc użytkownik zobaczy na ekranie animację. Jej szybkością można sterować zmieniając wartość argumentu funkcji `halfdelay()`. Po zakończeniu pętli usuwane jest okno planszy gry za pomocą wywołania funkcji `delwin()`, a następnie kończona jest praca z biblioteką `curses` poprzez wywołanie `endwin()`. Wyniki działania obu funkcji są sprawdzane celem wykrycia wystąpienia potencjalnych wyjątków. Ponieważ macierz jest rozmiaru 32×32, to aby program zdziałał terminal musi być tak skonfigurowany, aby miał co najmniej 32 wiersze.

Podziękowania

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

Pytania

?

KONIEC

Dziękuję Państwu za uwagę.