
Podstawy programowania.

Wykład 12

Funkcje raz jeszcze.

Biblioteka standardowa.

Wskaźniki do funkcji

- W maszynie von Neumanna kod i dane są przechowywane w tej samej adresowalnej pamięci
- Każda funkcja zapisana w kodzie binarnym jest przechowywana w pewnym obszarze pamięci o określonym adresie
- W programach C można posługiwać się wskaźnikami do funkcji, zawierającymi adres początku funkcji
 - najczęściej wskaźniki do funkcji wykorzystuje się w roli argumentów przesyłanych do funkcji
 - przykład wywołania: funkcja rysująca wykresy funkcji matematycznych w przedziale (a,b):
`rysuj_wykres(sinus, a, b);`
 - wywołanie zawierające wskaźnik do funkcji jako argument jest proste – rolę wskaźnika pełni nazwa funkcji
 - w podanym przykładzie `sinus` jest gdzieś zdefiniowaną funkcją

Wskaźniki do funkcji

→ Deklaracje wskaźników do funkcji:

- w deklaracji wskaźnika musi zostać zawarta cała informacja o typie funkcji, obejmująca typy argumentów i typ zwracanego wyniku
- w przykładzie rysowania wykresów deklaracja funkcji `rysuj_wykres` powinna mieć postać:

```
void rysuj_wykres(double (*funkcja)(double), double a, double b);
```

- deklaracją funkcji `sinus` zgodną z prototypem funkcji `rysuj_wykres` (i wykorzystaniem w przykładowym wywołaniu) jest

```
double sinus (double arg_d);
```

- standardowe użycie funkcji `sinus` ma postać

```
double y = sinus(x); // x jest typu double
```

- wewnątrz funkcji `rysuj_wykres`, która otrzymuje wskaźnik do funkcji, wywołanie przesłanej funkcji ma zawsze tę sama postać

```
double y = (*funkcja)(x);
```

- nazwa funkcji `sinus` (ani żadnej innej wskazywanej) nie pojawia się w funkcji `rysuj_wykres` (co najwyżej w komentarzach)

Wskaźniki do funkcji

Deklaracje w C mogą mieć złożony charakter, szczególnie jeśli zawierają wskaźniki do funkcji (przykłady z podręcznika K&R):

```
char **argv
    argv: pointer to pointer to char
int (*daytab)[13]
    daytab: pointer to array[13] of int
int *daytab[13]
    daytab: array[13] of pointer to int
void *comp()
    comp: function returning pointer to void
void (*comp)()
    comp: pointer to function returning void
char ((*x())[5])()
    x: function returning pointer to array[5] of
    pointer to function returning char
char ((*x[3])())[5]
    x: array[3] of pointer to function returning
    pointer to array[5] of char
```

Argumenty wywołania programu

- Funkcja *main* może posiadać argumenty (parametry) wejścia
- składnia definicji funkcji *main* jest ściśle określona
 - funkcja *main* albo nie przyjmuje żadnych argumentów

```
int main(void) { /* ... */ }
```
 - albo przyjmuje je w postaci

```
int main(int argc, char *argv[]) { /* ... */ }
```

 - każdorazowo `argv[argc]==NULL`
 - jeśli `argc` jest większe od zera oznacza to, że środowisko wywołania przekazuje argumenty (parametry) w napisach wskazywanych przez elementy tablicy `argv` (o długości `argc+1`)
 - » `argv[0]` jest zawsze nazwą programu (pliku binarnego)
 - jeśli `argc` jest większe od jeden oznacza to, że przy wywołaniu podane zostały argumenty wejścia, zamienione następnie przez środowisko wywołania w napisy

Argumenty wywołania programu

→ Przykładowy program obsługi parametrów wejścia (argumentów wiersza polecenia wywołania programu)

- najprostszy wariant wypisanie argumentów :

```
int main(int argc, char *argv[])  
// alternatywa: int main(int argc, char **argv)  
{  
    if(argc<=2) { printf( "Usage: ....\n"); exit(0); }  
    for (int i = 0; i < argc; i++) printf( " %s", argv[i]);  
    printf ("\n");  
}
```

→ Obsługa parametrów wejścia jest zależna od programu i polega na wykonywaniu odpowiednich operacji na napisach

- można do tego wykorzystywać funkcje operujące na napisach zawarte w bibliotece standardowej C

Funkcje wplatane (*inline*)

- Kompilator napotykając funkcję zdefiniowaną jako *inline* umieszcza treść tej funkcji tam, gdzie jest ona wywoływana
 - operacja jest nazywana wplataniem lub rozwinięciem w miejscu wywołania
 - oszczędza to w trakcie wykonania narzutu związanego z wywołaniem funkcji (m.in. obsługa stosu)
- Funkcje wplatane (*inline*) powinny być stosowane dla funkcji, których treść jest bardzo krótka i które wywoływane są w programie wiele razy
- Ostateczną decyzję o tym czy funkcja zostanie wpleciona w miejscu wywołania podejmuje kompilator
 - słowo kluczowe *inline* jest tylko sugestią dla kompilatora
- Jeśli funkcja jest oznaczona jako *inline*, to jej definicja musi być dostępna dla kompilatora w trakcie przetwarzania pliku źródłowego
 - naturalnym miejscem na definiowanie funkcji wplatanych wykorzystywanych w wielu plikach źródłowych są pliki nagłówkowe

Tworzenie bibliotek

- Biblioteki zawierają skompilowane funkcje gotowe do wykorzystania w programach
- Dwa podstawowe typy bibliotek to:
 - biblioteki statyczne
 - zawierają zarchiwizowane pliki pośrednie kodu źródłowego
 - pliki bibliotek statycznych są linkowane przy tworzeniu plików wykonywalnych
 - biblioteki dynamiczne
 - są dołączane w trakcie uruchomienia programu, pozwalają na zmniejszenie rozmiaru kodu wykonywalnego i mniejsze wymagania pamięciowe, w sytuacji wykonywania wielu programów korzystających z tych samych bibliotek
 - szczegóły tworzenia i wykorzystania bibliotek dynamicznych są zależne od systemu
 - programy wykonywalne stosujące biblioteki dynamiczne są mniej przenośne niż programy zawierające zlinkowane biblioteki statyczne

Biblioteka standardowa C

- Praktycznie każdy język programowania dostarcza zestaw funkcji wspomagających tworzenie kodu
 - zestaw funkcji określany przez specyfikację (standard) języka jest nazywany biblioteką standardową języka
 - niektóre z funkcji są konieczne do pisania programów
 - np. funkcje obsługi wejścia/wyjścia
 - inne ułatwiają pisanie przez rozwiązanie standardowych problemów programistycznych
 - np. funkcje obsługi napisów, funkcje matematyczne
- Biblioteka standardowa C jest relatywnie mała (w porównaniu z np. rozbudowanymi bibliotekami języków obiektowych)
 - niemniej jednak definiuje kilkaset funkcji, typów danych i makrodefinicji
 - omówienie całej biblioteki standardowej byłoby niezwykle obszerne
 - wiele z definiowanych elementów ma ograniczone zastosowanie

Biblioteka standardowa C

→ Pliki nagłówkowe biblioteki standardowej C

<assert.h>

<complex.h>

<ctype.h>

<errno.h>

<fenv.h>

<float.h>

<inttypes.h>

<iso646.h>

<limits.h>

<locale.h>

<math.h>

<setjmp.h>

<signal.h>

<stdalign.h>

<stdarg.h>

<stdatomic.h>

<stdbool.h>

<stddef.h>

<stdint.h>

<stdio.h>

<stdlib.h>

<stdnoreturn.h>

<string.h>

<tgmath.h>

<threads.h>

<time.h>

<uchar.h>

<wchar.h>

<wctype.h>

Biblioteka standardowa C – <stdio.h>

- Najważniejszą grupą funkcji biblioteki standardowej są funkcje obsługi wejścia/wyjścia
- Urządzeniami wejścia/wyjścia mogą być pliki lub inne elementy traktowane przez język jako strumienie
 - znaków – podzielone na linie
 - binarne – jako sekwencja znaków (bajtów)
 - końcem strumienia jest zawsze znak końca pliku EOF
- Standardowa biblioteka wejścia/wyjścia rozróżnia strumienie zawierające standardowe znaki jednobajtowe i znaki z zestawu rozszerzonego (*wide characters*)
 - do obsługi różnych typów strumieni przeznaczone są różne funkcje
 - omawiamy tylko funkcje obsługi strumieni znaków jednobajtowych
 - większość funkcji posiada swoje odpowiedniki dla strumieni znaków rozszerzonych

Biblioteka standardowa C – `<stdio.h>`

→ C posługuje się pojęciem standardowego wejścia i standardowego wyjścia (*standard input* – *stdin*, *standard output* – *stdout*)

- najczęściej są nimi klawiatura i ekran terminala
- wykorzystanie systemowych narzędzi przekierowania strumieni wejścia i wyjścia pozwala na używanie plików w miejsce standardowego wejścia i wyjścia
- standard definiuje także trzeci domyślny strumień – *standard error*, *stderr*
- *stderr* służy do wypisywania komunikatów diagnostycznych
 - przykład przekierowania standardowego wyjścia z zastąpieniem pliku:

```
ls /home/* > output.txt
```

- przykład przekierowania standardowego wejścia i wyjścia (*tcsh*):

```
ls < input.txt > output.txt
```

- przykład przekierowania standardowego wyjścia i *stderr* do różnych plików z dopisaniem zawartości zamiast zastąpienia zawartości (*bash*):

```
ls /home/* >> output.txt 2>> error.txt
```

- przykład przekierowania standardowego wyjścia i *stderr* do tego samego pliku z dopisaniem zawartości zamiast zastąpienia zawartości (*bash*):

```
ls /home/* &>> log.txt
```

- itd. itp. (różne warianty dla różnych powłok)

- wariantem powyższych manipulacji jest często stosowane przekierowywanie standardowego wyjścia jednego z poleceń na standardowe wejście innego:

```
ls /home/* | more
```

Biblioteka standardowa C – `<stdio.h>`

- Podstawowe funkcje obsługi standardowego wejścia i wyjścia:
- `int getchar(void)` – pobranie pojedynczego znaku z *stdin*
 - `int putchar(int)` – przesłanie pojedynczego znaku do *stdout*
 - `int printf (char *format, arg1, arg2, ...)` – formatowane wyjście
 - `int scanf (char *format, arg1, arg2, ...)` – formatowane wejście
 - szczegółowe zasady formatowania wejścia/wyjścia za pomocą odpowiedniej składni napisu *format* można znaleźć w wielu źródłach (standard, podręczniki, źródła internetowe)
 - obie funkcje formatowanego wejścia/wyjścia korzystają z oferowanej przez C możliwości definiowania funkcji o zmiennej liczbie argumentów
 - do obsługi wywołań takich funkcji należy używać makrodefinicji zawartych w pliku nagłówkowym `<stdarg.h>`
 - istnieją funkcje obsługi pojedynczych linii – `gets` i `puts`
 - do obsługi plików oraz napisów (jako strumieni znaków) istnieją specjalne funkcje, często będące wariantami funkcji do obsługi standardowego wejścia i standardowego wyjścia (np. `fprintf` i `fscanf` do obsługi plików oraz `sprintf` i `sscanf` do obsługi napisów)

Biblioteka standardowa C – `<stdio.h>`

- Program w C wiąże strumień wejścia/wyjścia z konkretnym plikiem poprzez operację otwarcia pliku
 - pojedynczy program może jednocześnie posiadać wiele otwartych plików
 - pojedynczy plik może być jednocześnie otwarty przez wiele programów
 - system operacyjny musi zagwarantować sposoby bezpiecznego i poprawnego korzystania z plików w takich sytuacjach
 - połączenie z plikiem ustaje po operacji zamknięcia pliku
- Interfejs obsługi plików:
 - otwarcie pliku
`FILE *fopen(const char * restrict name, const char * restrict mode);`
 - napis określający tryb interakcji z plikiem może przybierać wartości:
 - "r" – otwarcie do odczytu
 - "w" – otwarcie do zapisu (utworzenie lub wyzerowanie długości)
 - "a" – otwarcie lub utworzenie pliku do zapisu na końcu (*append*)
 - istnieje szereg innych wariantów (np. dla plików binarnych)
- zamknięcie pliku: `int fclose(FILE *fp)`

Biblioteka standardowa C – `<stdio.h>`

→ Reprezentacją otwartego pliku w programie C jest wskaźnik do pliku

`FILE *fp;` // makro `FILE` jest zdefiniowane w `<stdio.h>`

- procedury systemowe obsługi plików (*open, creat, close, lseek, read, write*), które mogą być wykorzystywane w programach C, korzystają z pojęcia deskryptora pliku, reprezentowanego przez liczbę, a nie przez wskaźnik
- Przykłady wykorzystania plików w programach C:
 - otwarcie pliku *moj_plik.txt* w trybie do zapisu i użycie *fprintf*:

```
FILE *fp;  
fp = fopen("moj_plik.txt", "w");  
fprintf(fp, "%f\n", 3.14);  
fclose(fp);
```
 - otwarcie pliku *moj_plik.txt* w trybie do odczytu i użycie *fscanf*:

```
FILE *fp = fopen("moj_plik.txt", "r");  
float pi_zpliku; fscanf(fp, "%f", &pi_zpliku);  
fclose(fp);
```

Biblioteka standardowa C – <stdio.h>

→ Interfejs obsługi plików:

- funkcje wejścia/wyjścia dla plików analogiczne jak dla standardowego wejścia i wyjścia
 - `int fgetc(FILE *fp)` – pobranie znaku z pliku
 - `int ungetc(int c, FILE *fp)` – cofnięcie odczytu znaku z pliku
 - plik nie jest zmieniany, ale kolejne `fgetc` odczyta zwrócony znak
 - `int fputc(int c, FILE *fp)` – zapis znaku do pliku
 - `char *fgets(char *line, int maxline, FILE *fp)` – odczyt linii z pliku
 - `maxline` jest rozmiarem tablicy (łącznie z kończącym `'\0'`)
 - » `fgets` wczytuje co najwyżej `maxline-1` znaków
 - `int fputs(char *string, FILE *fp)` – zapis napisu do pliku
 - `int fscanf(FILE *fp, char *format, arg1, ...)` - formatowane wejście
 - `int fprintf(FILE *fp, char *format, arg1, ...)` - formatowane wyjście
 - dla każdej z powyższych funkcji można `fp` zamienić na `stdin`, `stdout` lub `stderr`

Biblioteka standardowa C – <stdio.h>

→ Interfejs obsługi plików:

- wybrane pozostałe funkcje obsługi wejścia/wyjścia dla plików

`size_t fread(void * ptr, size_t size, size_t nmemb, FILE * stream);`

- odczyt *nmemb* elementów o rozmiarze *size* z **stream* do **ptr*

`size_t fwrite(const void * ptr, size_t size, size_t nmemb, FILE * stream);`

- zapis *nmemb* elementów o rozmiarze *size* z **ptr* do **stream*

`int fseek(FILE *stream, long int offset, int whence);`

- ustawienie pozycji (dla kolejnego odczytu lub zapisu) w pliku **stream* w odległości *offset* od punktu wskazywanego przez *whence*

‣ *whence* może być określone jako:

‣‣ SEEK_SET – początek pliku

‣‣ SEEK_CUR – aktualna pozycja w pliku

‣‣ SEEK_END – koniec pliku

`long int ftell(FILE *stream)` - dla plików binarnych zwraca liczbę bajtów od początku do obecnej pozycji w pliku

Biblioteka standardowa C – <stdio.h>

- Obsługa napisów za pomocą *sprintf* i *scanf*
- funkcje formatowanego wyjścia i wejścia *sprintf* i *scanf* są analogiczne do funkcji *fprintf* i *fscanf* operujących na plikach
 - argumentem zamiast wskaźnika do pliku jest napis
 - `int sprintf (char * napis, char *format, arg1, arg2, ...)` – formatowane wyjście
 - funkcji *sprintf* można użyć do przekształcania danych liczbowych w napisy:

```
char linia_plus[MAXLINE+20]; // miejsce na numer linii
sprintf(linia_plus, "linia numer %2d: \"%s\"\n", nr_linii, linia);
```
 - `int scanf (char * napis, char *format, arg1, arg2, ...)` – formatowane wejście
 - źródłem danych jest tablica (napis), zwracaną wartością (podobnie jak dla *scanf* i *fscanf*) jest liczba wczytanych argumentów
 - argumenty zostają wczytane tylko w przypadku zgodności formatu z danymi wejściowymi (w tablicy dla *scanf*, *stdin* dla *scanf* i pliku dla *fscanf*)

Biblioteka standardowa C – `<assert.h>`

- Plik nagłówkowy `<assert.h>` definiuje makro `assert` służące do sprawdzania spełnienia warunków w trakcie wykonania programu i podstawowej obsługi błędów
- Przykład zastosowania plikowego wejścia/wyjścia i makra `assert`

```
char napis[100];  
FILE *fp = fopen("nazwa_pliku", "r");  
assert(fp != NULL);  
fgets(napis, 100, fp);  
printf("Przeczytane: %s\n", napis);  
fclose(fp);
```

- W przypadku kiedy plik `"nazwa_pliku"` nie istnieje w katalogu roboczym, `fopen` zwraca `NULL` a makro `assert` przerywa wykonanie programu (przez wywołanie `abort`) po wydrukowaniu komunikatu o błędzie, np:
`zmienne: zmienne.c:192: main: Assertion `fp!=((void *)0)' failed.`
- Sprawdzenie makra `assert` można wyłączyć przez zdefiniowanie makra `NDEBUG` (np. przekazując opcję `-DNDEBUG` w trakcie kompilacji)

Biblioteka standardowa C - <string.h>

→ Podstawowe funkcje obsługi napisów (przypomnienie)

`memcpy(void * s1, void * s2, size_t n)` - przepisanie n bajtów z s1 do s2

- ustalaniu wartości służy funkcja `memset(void *s, int c, size_t n)`;

`strcat(s,t)` – dopisanie t na koniec s

`strncat(s,t,n)` – to samo dla n pierwszych znaków t

`strcmp(s,t)` – porównanie dwóch napisów

`strncmp(s,t,n)` – to samo dla n pierwszych znaków

`strcpy(s,t)` – skopiowanie t do s

`strncpy(s,t,n)` – to samo dla co najwyżej n znaków

`strlen(s)` – obliczenie długości napisu (bez \0)

- funkcje zaznaczone na czerwono mogą nie być bezpieczne, lepiej używać bezpiecznych wariantów z ograniczeniem liczby znaków na których operuje funkcja
 - niebezpieczeństwo polega na przekroczeniu rozmiaru (przepełnieniu) buforów, w których przechowywane są napisy

Biblioteka standardowa C - `<stdlib.h>`

- Plik nagłówkowy *stdlib.h* definiuje wiele wspomagających funkcji użytkowych, z których niektóre były już omawiane w trakcie wykładów (i wykorzystywane w trakcie laboratoriów)
- Przykłady:
 - dynamiczne zarządzanie pamięcią: *malloc, calloc, alloc, free* (i inne)
 - zamiana napisów na liczby: *atoi, atof, strtod, strtol* (i inne)
 - generacja liczb losowych: *rand, srand*
 - funkcja wykonania polecenia systemowego: *system* (np. *system("ls");*)
 - sortowanie szybkie i wyszukiwanie binarne: *qsort* i *bsearch*
 - wartość bezwzględna dla liczb całkowitych: *abs*
 - funkcje obsługi przerwania wykonywania programu: *abort, exit* (i inne)
 - *void abort(void)* – zakończenie "nienormalne" (sygnał SIGABRT)
 - *void exit(int status)* – zakończenie normalne, zwrócenie wartości do środowiska wykonania

Biblioteka standardowa C - <stdlib.h>

→ zamiana liczb zapisanych znakami na zmienne liczbowe:

```
int atoi(const char *nptr);
```

```
long int strtol(const char *nptr, char **endptr, int base);
```

```
double atof(const char *nptr);
```

```
double strtod(const char *nptr, char **endptr);
```

→ sortowanie szybkie:

```
void qsort(void *base, size_t nmemb, size_t size,  
           int (*compar)(const void *, const void *));
```

→ wyszukiwanie binarne:

```
void *bsearch(const void *key, const void *base,  
             size_t nmemb, size_t size,  
             int (*compar)(const void *, const void *));
```

Biblioteka standardowa C - <ctype.h>

→ Operacje na znakach:

- funkcje sprawdzające znak, zwracające 0 w przypadku odpowiedzi negatywnej:

`isalpha(c)` – czy znak jest literą

`isupper(c)` – czy znak jest wielką literą

`islower(c)` – czy znak jest małą literą

`isdigit(c)` – czy znak jest cyfrą

`isalnum(c)` – czy znak jest znakiem alfanumerycznym

`isspace(c)` – czy znak jest odstępem, znakiem tabulacji, znakiem nowej linii

- funkcje konwersji znaków:

`toupper(c)` – z małej do wielkiej litery

`tolower(c)` – z wielkiej do małej litery

- oraz kilka innych (*`isblank`*, *`isctrl`*, *`isgraph`*, *`isprint`*, *`ispunct`*, *`isxdigit`*)

Biblioteka standardowa C - <math.h>

- Wybrane funkcje matematyczne
 - zawsze argumenty i zwracana wartość są typu double
 - `sin(x)` - sinus
 - `cos(x)` - cosinus
 - `atan2(y,x)` – arcus tangens x/y
 - `exp(x)` – funkcja wykładnicza (exponencjalna)
 - `log(x)` – logarytm naturalny ($x > 0$)
 - `log10(x)` - funkcja logarytmiczna o podstawie 10 ($x > 0$)
 - `pow(x,y)` - x^y
 - `sqrt(x)` – pierwiastek kwadratowy ($x \geq 0$)
 - `fabs(x)` – wartość bezwzględna
 - funkcje sprawdzania argumentów
 - np *isfinite*, *isnan*
 - oraz kilkadziesiąt innych funkcji

Biblioteka standardowa C

- Najważniejsze inne grupy funkcji i makrodefinicji
 - `<complex.h>` - obsługa zmiennych zespolonych
 - `<errno.h>` - obsługa błędów
 - `<limits.h>` - rozmiary zmiennych typu całkowitego
 - `<inttypes.h>` - konwersje między typami całkowitymi
 - `<locale.h>` - funkcje, typy i makra dotyczące lokalizacji
 - `<setjmp.h>` - specjalny powrót z wywołania w przypadku błędu
 - `<signal.h>` - obsługa sygnałów przesyłanych między procesami
 - `<stdbool.h>` - makra do obsługi zmiennych boolowskich
 - `<stddef.h>` - m.in. definicje *size_t*, *ptrdiff_t*, *NULL*
 - `<stdint.h>` - definicje specjalnych typów całkowitych
 - `<stdatomic.h>`, `<threads.h>` - dla programów wielowątkowych
 - `<time.h>` - obsługa czasu i daty
 - `<uchar.h>`, `<wchar.h>` - rozszerzone zestawy znaków