

# Informatyka 2

Politechnika Białostocka - Wydział Elektryczny  
Elektrotechnika, semestr III, studia stacjonarne I stopnia  
Rok akademicki 2018/2019

## Wykład nr 6 (06.11.2018)

dr inż. Jarosław Forenc

## Plan wykładu nr 6

- Pamięć a zmienne w programie
  - zmienne automatyczne (auto)
  - zmienne rejestrowe (register)
  - zmienne zewnętrzne (extern)
  - zmienne statyczne (static)
  - struktura procesu w pamięci komputera, ramka stosu
- Programy wielomodułowe
- Operacje wejścia-wyjścia w języku C
  - typy standardowych operacji wejścia wyjścia
  - strumienie
  - standardowe strumienie: stdin, stdout, stderr

## Pamięć a zmienne w programie

- Ze względu na czas życia wyróżnia się w programie:
  - **obiekty statyczne** - istnieją od chwili rozpoczęcia działania programu aż do jego zakończenia
  - **obiekty dynamiczne** - tworzone i usuwane z pamięci w trakcie wykonania programu
    - automatycznie (bez udziału programisty)
    - kontrolowane przez programistę
- O typie obiektu (**statyczny** lub **dynamiczny**) decyduje klasa pamięci obiektu (ang. storage class)
  - **auto** - zmienne automatyczne
  - **register** - zmienne umieszczane w rejestrach procesora
  - **extern** - zmienne zewnętrzne
  - **static** - zmienne statyczne

## Zmienne automatyczne - auto

- Miejsce deklaracji: najczęściej początek bloku funkcyjnego ograniczonego nawiasami klamrowymi { } }
- Pamięć przydzielana automatycznie przy wejściu do bloku i zwalniana po wyjściu z niego
- Zakres widzialności: ograniczony do bloku, w którym zmienne zostały zadeklarowane (**zmienne lokalne**)
- Dostęp do zmiennych z innych bloków możliwy przez wskaźnik
- Jeśli zmienne są inicjalizowane, to odbywa się ona przy każdym wejściu do bloku, w którym zostały zadeklarowane
- Nie ma potrzeby jawnego używania **auto**, gdyż domyślnie zmienne wewnątrz bloków funkcyjnych są lokalne

```
auto int x;
```

## Zmienne rejestrowe - register

- Zazwyczaj o miejscu umieszczenia zmiennej automatycznej decyduje kompilator:
  - pamięć operacyjna - wolniejszy dostęp
  - rejestry procesora - szybszy dostęp
- Programista może zasugerować kompilatorowi umieszczenie określonej zmiennej automatycznej w rejestrach procesora
- Najczęściej dotyczy to zmiennych:
  - często używanych
  - takich, dla których czas dostępu jest bardzo ważny

```
register int x;
```

## Zmienne zewnętrzne - extern

- Miejsce deklaracji: poza blokami funkcyjnymi, najczęściej na początku pliku z kodem źródłowym
- Pamięć na zmienne jest przydzielana, gdy program rozpoczyna pracę i zwalniana, gdy program kończy się
- Zakres widzialności: globalny - od miejsca deklaracji do końca pliku z kodem źródłowym (**zmienne globalne**)
- Jeśli inna zmienna lokalna, ma taką samą nazwę jak globalna, to lokalna przesłania widoczność zmiennej globalnej
- W większości implementacji języka C zmienne **extern** są automatycznie inicjalizowane zerem
- Etykieta **extern** może być pominięta (chyba, że program składa się z kilku plików z kodem źródłowym)
- Zalecane jest ograniczenie stosowania zmiennych globalnych

## Zmienne statyczne - static

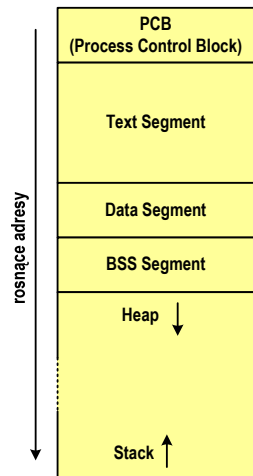
- Miejsce deklaracji: w bloku funkcyjnym jako automatyczne lub poza blokami funkcyjnymi, jako globalne
- Istnieją przez cały czas wykonywania programu, nawet po zakończeniu bloku funkcyjnego, w którym zostały zadeklarowane
- Zakres widzialności: zależy od sposobu deklaracji (automatyczne lub globalne)
- Zmienne **static** są automatycznie inicjalizowane zerem
- Mogą być inicjalizowane podczas deklaracji (tylko stałą wartością), inicjalizacja jest wykonywana tylko raz, podczas kompilacji programu

```
static int x = 10;
```

## Klasy pamięci zmiennych

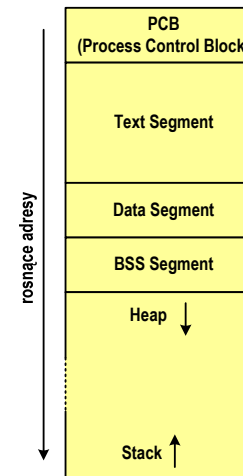
```
int a; /* extern int a; - zmienna globalna */  
void foo();  
  
int main(void)  
{  
    int b; /* auto int b; - zmienna lokalna */  
    register float a; /* zmienna automatyczna, rejestrowa */  
    foo(); foo(); foo();  
    return 0;  
}  
  
void foo()  
{  
    static int c = 1; /* zmienna statyczna */  
    {  
        double a; /* zmienna lokalna */  
    }  
    c++;  
}
```

## Struktura procesu w pamięci komputera



- **PCB** - blok kontrolny procesu
  - obszar pamięci operacyjnej zarezerwowany przez system operacyjny do zarządzania procesem
- **Text Segment**
  - kod programu czyli instrukcje w postaci binarnej
- **Data Segment**
  - zmienne globalne i statyczne zainicjalizowane niezerowymi wartościami
- **BSS Segment** (Block Started by Symbol)
  - zmienne globalne i statyczne domyślnie zainicjalizowane zerowymi wartościami

## Struktura procesu w pamięci komputera



- **Heap** - sterta
  - obszar zmiennych dynamicznych
  - pamięć w obszarze sterty przydzielana jest funkcjami `calloc()` i `malloc()`
- **Stack** - stos
  - zmienne lokalne (automatyczne)
  - parametry funkcji i adresy powrotu z funkcji (stack frame)

## Zmienne w pamięci komputera

```

int a;                               /* BSS Segment */
void foo();

int main(void)
{
    int b;                             /* Stack */
    float *a;                          /* Stack */
    a = (float *) malloc(400);          /* Heap - 400 bajtów */
    return 0;
}

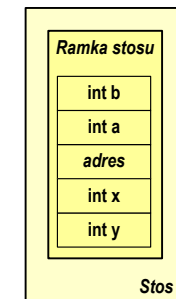
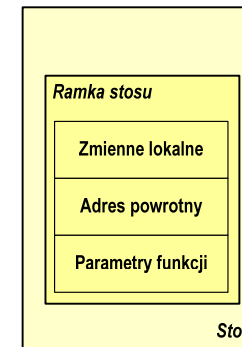
void foo()
{
    static int c = 1;                  /* Data Segment */
    {
        double a;                     /* Stack */
    }
    c++;
}
    
```

## Ramka stosu (stack frame)

- Każde wywołanie funkcji powoduje odłożenie na stosie tzw. **ramki stosu**

```

void fun(int x, int y)
{
    int a, b;
}
    
```



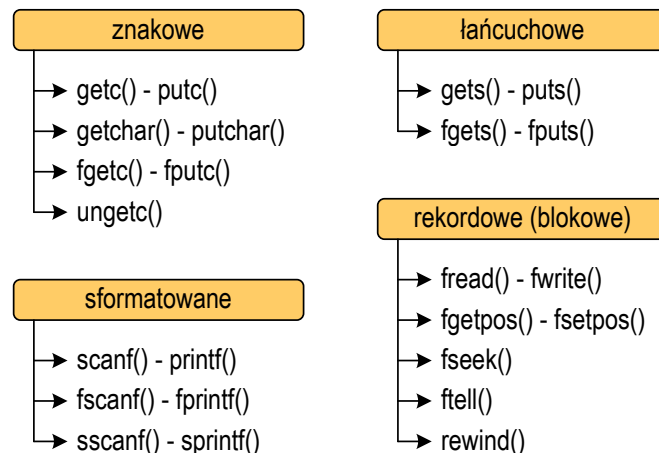
## Programy wielomodułowe

(Przykład w Visual C++ 2008)

## Operacje wejścia-wyjścia w języku C

- Operacje wejścia-wyjścia nie są elementami języka C
- Zostały zrealizowane jako funkcje zewnętrzne, znajdujące się w bibliotekach dostarczanych wraz z kompilatorem
- **Standardowe** wejście-wyjście (strumieniowe)
  - plik nagłówkowy **stdio.h**
  - duża liczba funkcji, proste w użyciu
  - ukrywa przed programistą szczegóły wykonywanych operacji
- **Systemowe** wejście-wyjście (deskryptorowe, niskopoziomowe)
  - plik nagłówkowy **io.h**
  - mniejsza liczba funkcji
  - programista sam obsługuje szczegóły wykonywanych operacji
  - funkcje bardziej zbliżone do systemu operacyjnego - działają szybciej

## Typy standardowych operacji wejścia-wyjścia



## Strumienie

- Standardowe operacje wejścia-wyjścia opierają się na **strumieniach** (ang. **stream**)
- Strumień jest pojęciem abstrakcyjnym - jego nazwa bierze się z analogii między przepływem danych, a np. wody
- W strumieniu dane płyną od źródła do odbiorcy
- Użytkownik określa źródło i odbiorcę, typ danych oraz sposób ich przesyłania
- Strumień może być skojarzony ze zbiorem danych znajdujących się na dysku (plik) lub zbiorem danych pochodzących z urządzenia znakowego (klawiatura)
- Niezależnie od fizycznego medium, z którym strumień jest skojarzony, wszystkie strumienie mają podobne właściwości

## Strumienie

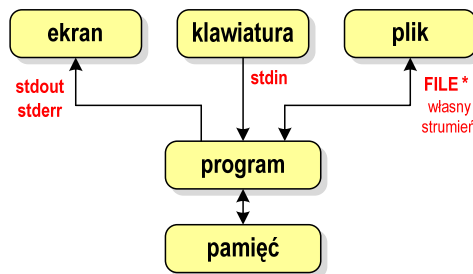
- Strumienie reprezentowane są przez zmienne będące wskaźnikami na struktury typu **FILE** (definicja w pliku **stdio.h**)

```
struct _iobuf {
    char *_ptr;
    int _cnt;
    char *_base;
    int _flag;
    int _file;
    int _charbuf;
    int _bufsiz;
    char *_tmpfname;
};
typedef struct _iobuf FILE;
```

- Podczas pisania programów nie ma potrzeby bezpośredniego odwoływania się do pól tej struktury

## Strumienie

- Współpraca programu z „otoczeniem”



- Istnieją funkcje wejścia-wyjścia:
  - korzystające domyślnie z określonego strumienia
  - wymagające podania strumienia

## Strumienie

- W każdym programie automatycznie tworzone są i otwierane trzy standardowe strumienie wejścia-wyjścia:
  - **stdin** - standardowe wejście, skojarzone z klawiaturą
  - **stdout** - standardowe wyjście, skojarzone z ekranem monitora
  - **stderr** - standardowe wyjście dla komunikatów o błędach, skojarzone z ekranem monitora

```
_CRTIMP FILE * __cdecl __iob_func(void);
#define stdin (&__iob_func()[0])
#define stdout (&__iob_func()[1])
#define stderr (&__iob_func()[2])
```

- Funkcja **printf()** niejawnie używa strumienia **stdout**
- Funkcja **scanf()** niejawnie używa strumienia **stdin**

## Koniec wykładu nr 6

Dziękuję za uwagę!